PCAPAC 2022
OCT 4-7

13th International Workshop
on Emerging Technologies
and Scientific Facilities Controls

PROCEEDINGS

PCaPAC 2022 hosted by ELI Beamlines
in Dolní Břežany/Prague, Czech Republic
4-7 October 2022

PCAPAC
OCT 4-7 2022

Venue

ELI Beamlines
Za Radnicí 835
25241 Dolní Břežany
Czech Republic

Information on how to get to the ELI Beamlines can be found on the website.

- Registration desk - in the entrance foyer on the ground floor
- Conference & Proceedings room – on the first floor
- Poster session, exhibition, catering – in the Atrium on the ground floor

Registration on site

- Tuesday – October 4, 2022 / from 8:30
- Wednesday – October 5, 2022 / from 8:30

All participants are asked to wear their **name badges** throughout the conference, in particular for the conference social programme and coffee breaks, lunches, and welcome party.

Payments on site
There will not be any possibility to pay in cash or by credit cards on site. Online payment by credit cards are possible at any time.

Programme

The conference programme is continuously updated and is available at
`https://indico.eli-beams.eu/event/360/page/405-programme`

Instructions for authors

Poster session
There will be self-standing "carpet" boards prepared for all poster presentations. The conference organizers will provide necessary materials for poster mounting - pins, or you can use your own velcro straps

Poster table size:
- 1 meter wide
- 2 meters high

Please note, that it is not possible to print out your poster on site.

Social programme

Welcome Party
Tuesday - October 4, 2022 / 17:00 - 19:00

The welcome dinner will take place in the Atrium of ELI Beamlines.
We thank D-TACQ Solutions Ltd. for the sponsoring of this event.

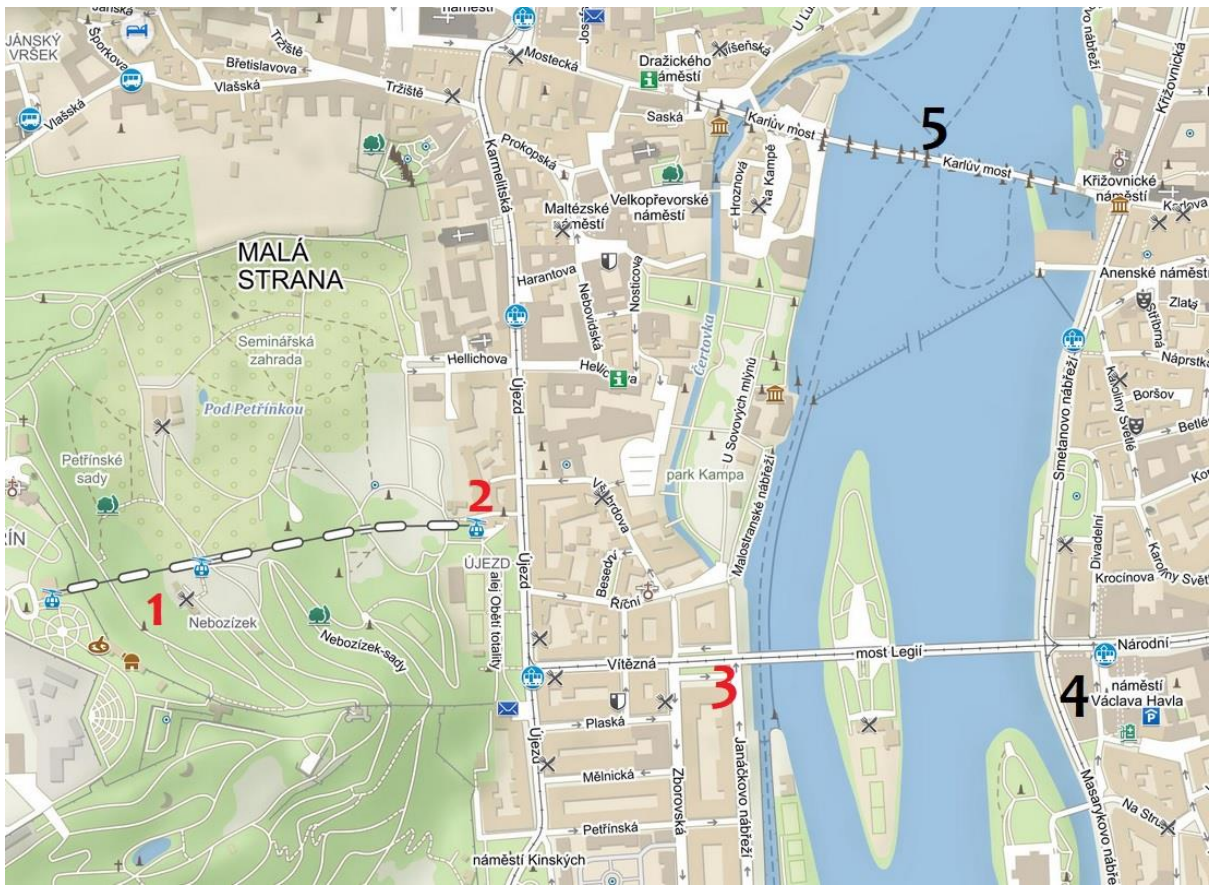
Dinner

Thursday – October 6, 2022 / 18:30-21:00

The conference dinner will be held in the Nebozízek Restaurant located in Prague.
The dinner will be served as a buffet.

How to reach the dinner?

- There will be a bus provided for transfer from ELI Beamlines, departing at 17:30.
- It will take you downtown close to "*Most Legii*" (3, see the map).
- Then continue for 5 minutes on foot to the funicular/cable car (2).
- It's necessary to stamp a ticket when entering. Only longer-term (24 hours or more) tickets are valid, and you will receive an one-day ticket for this case. It will be inserted in your name badge. The ticket is valid 24 hours from the time of marking.
- Get off at the "*Nebozízek*" station (1).
- No bus after the dinner is provided; the event ends there.
  You will be close to the city centre and public transport is in walking distance.



Legend:
1 = Nebozízek restaurant / 2 = funicular / 3 = exit from a bus
4 = National Theatre / 5 = Charles Bridge

International Program Committee (IPC)

Michael Abbott	Diamond	UK
Yves-Marie Abiven	SOLEIL	France
Reinhard Bacher	DESY	Germany
Yung-Sen Cheng	NSRRC	Taiwan
Mark Clift	Australian Synchrotron	Australia
Guifre Cuni	ALBA	Spain
Philip Duval	DESY	Germany
Wolfgang Mexner	KIT/IBPT	Germany
Tim Mooney	ANL/APS	USA
Norihiko Kamikubota	KEK-Tokai	Japan
Takashi Kosuge	KEK-Tsukuba	Japan
James Rezende Piton	LNLS	Brazil
Mark Plesko	Cosylab	Slovenia
Fatnani Pravin	RRCAT	India
Anindya Roy	VECC	India
Takashi Sugimoto	SPring8/SACLA	Japan
Alessandro Stecchi	INFN	Italy
Glen Wright	CLS	Canada
Yingbing Yan	SSRF/SXFEL/SHINE	China

Local Organizing Committee (LOC)

Birgit Plötzeneder	**Chair**	ELI Beamlines	Czech Republic
Ana tajminger		ELI Beamlines	Czech Republic
Purbaj Pant		ELI Beamlines	Czech Republic
Romana Koová		ELI Beamlines	Czech Republic

JACoW Editorial Team (JET)

Ana tajminger	**Editor-in-Chief**	ELI Beamlines	Czech Republic
Volker RW Schaa	**Technical Editor**	GSI	Germany

Contents

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE012

THE TECHNICAL DESIGN CONCEPT OF THE NEW ACCELERATOR CONTROL SYSTEM FOR PETRA IV

R. Bacher, T. Delfs, T. Tempel, T. Wilksen
Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract

At DESY, a technical design report is currently being prepared for the upgrade of the PETRAIII synchrotron light source to PETRAIV, a fourth-generation low-emittance machine. Within the framework of this planned project, the accelerator control system is also to be renewed and adapted to the growing user requirements. This concerns on the one hand the core components of the control system itself, but also the hardware interfaces, the technical and beam physics control applications, the data acquisition and archiving systems, and the entire supporting IT infrastructure. The paper reports on the details of the proposed technical design concept.

INTRODUCTION

With PETRA III, DESY operates one of the best storage ring X-ray radiation sources in the world. PETRA III is a 2300-metre-long storage ring feeding 24 user beamlines. It is operated either in brightness mode (480 equally distributed bunches, 120 mA stored beam) or in timing mode (40 equally distributed bunches, 100 mA stored beam), latter a unique feature of PETRAIII. Research groups from all over the world use the particularly brilliant, intense X-ray light for a variety of experiments - from medical to materials research.

DESY plans to expand it into an ultimate, high-resolution 3D X-ray microscope for chemical and physical processes. PETRA IV [1] will extend the X-ray view to all length scales, from the atom size to millimetres. Researchers can thus analyse processes inside a catalyst, a battery or a microchip under realistic operating conditions and specifically tailor materials with nanostructures. PETRA IV offers outstanding possibilities and optimal experimental conditions for industry. Special emphasis is also placed on the sustainable operation of the facility.

PETRA IV will replace the PETRA III facility and will be housed by the existing PETRA III buildings. An additional experimental hall will provide space for additional 18 user beamlines. In addition, a new synchrotron (DESY IV) will serve as booster between the existing electron source LINAC II and PETRA IV. Recently, research has begun on the development and construction of a 6 GeV laser plasma injector.

In 2020, a preparatory phase for the future project PETRA IV was initiated with the aim of submitting a Technical Design Report by mid-2023. Construction work is scheduled to begin in early 2027, followed by a commissioning phase in 2029.

The following chapter will describe the technical design concept of the accelerator control system of the future PETRA IV facility.

TECHNICAL DESIGN CONCEPT

The development and implementation of the future PETRA IV accelerator control system will be embedded in a long-term process to consolidate and simplify the whole accelerator control system landscape at DESY and to take advantage of synergies between the accelerator facilities operated by DESY. Support and maintenance of the existing control system framework used at PETRA III will not be continued beyond its expected lifetime. Central control system services such as data acquisition and archiving or configuration management are to be renewed and prepared for future requirements.

The accelerator control system of PETRA IV will closely follow the control system concept implemented at the European XFEL which is pulsed linear accelerator and free-electron laser. Therefore, the control system concept of PETRAIV will be adapted where necessary to the special needs of a storage ring X-ray radiation source.

Control System Framework

The Distributed Object-Oriented Control System (DOOCS) [2] will form the basis of the future control system of PETRA IV (Fig. 1). DOOCS is the established control system framework at FLASH, European XFEL and other conventional accelerator facilities operated by DESY, as well as advanced accelerator projects based on plasma wake field acceleration. Design features of DOOCS include:

- DOOCS follows the object-oriented design paradigm. Devices and data are objects. The basic entity is a device server representing some control system hardware or logic.
- DOOCS is based on a distributed client-server architecture. Each control system parameter is made accessible via network calls. Its transportation layer is currently based on the standardized, industrial RPC protocol with XDR data representation. Integration of the 0MQ protocol is in progress.
- The DOOCS server core library and the server API are written in C++. A variety of fieldbus and hardware interfaces are supported via device classes. These are accessible through additional libraries which can be linked as needed individually to the server core library.
- A fine-grained mechanism for access authorization as well as monitoring and extensive logging functions are provided.
- Libraries for creating client applications in C++, Java, Python or MATLAB are available either as a separate implementation or as C-bindings.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE012

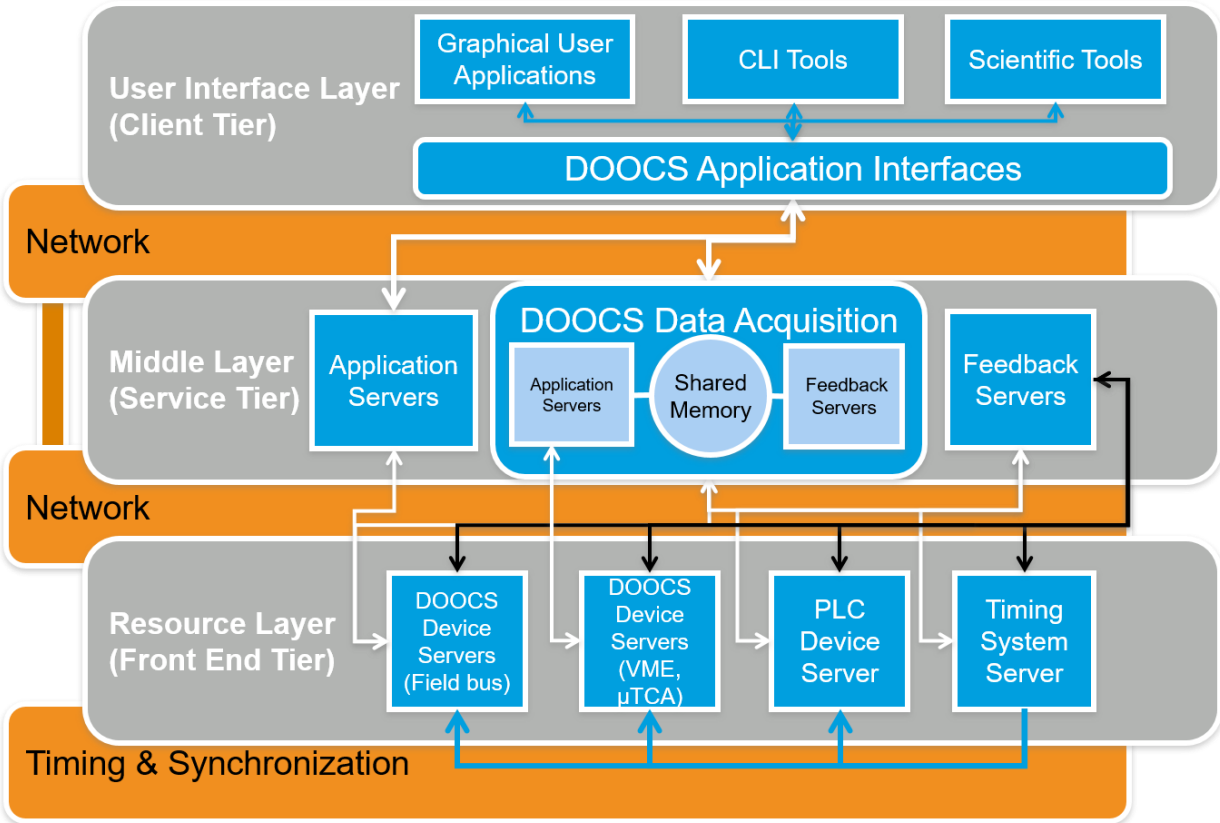


Figure 1: DOOCS control system layout.

- Through the client API DOOCS provides access to multiple popular control system such as EPICS and TANGO. At DESY, EPICS is used for facility control (electrical power and water distribution, ventilation and air conditioning) and control of the cryogenic systems, while TANGO is the standard control system for operating the beam line components and the experimental equipment.

The initial development of DOOCS dates back to 1993. Since that time, it has steadily developed into a powerful, reliable and versatile control system. Recently, a roadmap was established to meet the increasing user demands over the next decade and to continue to keep pace with the rapid developments in IT.

Graphical User Interfaces

The proven Java DOOCS Data Display (JDDD) [3] is chosen as standard user interface. JDDD is a Java application and follows a thin-client approach with a functional and rich set of widgets. Individual UI components can be easily created through a versatile editor IDE without the knowledge of any programming language

While JDDD is the tool of choice for the standard beam operation as well as operating technical accelerator devices and systems, Python is increasingly becoming the preferred programming language for rapid prototyping and visualization of scientific procedures and data.

Even if JDDD also provides a secure, HTML5-based Web interface, with the advent of modern Web standards such as Progressive Web Apps (PWA) other promising alternatives to design graphical user applications seem to be available. PWA are multi-platform, browser-based applications with a look-and-feel of versatile classical desktop applications. The potential of graphical user applications based on the React [4] JavaScript framework and the D3.js data visualization library [5] is currently investigated and first prototype applications are implemented and tested with respect to performance and user acceptance.

Hardware Interfaces

In general, the hardware interfaces for triggered, high-performance applications (e.g. beam diagnostics, injection/ejection system, feedback systems, timing/synchronization system, machine protection system, RF control) will be compliant with the high-end MTCA.4 technology [6]. MTCA.4 is the accepted long-term standard for the DESY accelerators and is enjoying growing popularity within the accelerator community and the related industry. The operating system for server hosts running within the MTCA.4 platform will be Linux.

The base configuration of a MTCA system includes a power-supply, a Management-Controller Hub (MCH), a CPU as an Advanced Mezzanine Card (AMC), a Timing System AMC, and optionally a Timing System Rear Transition Module (RTM) if required.

Based on the existing MTCA.4 timing module used at the European XFEL, a successor model is currently being developed at DESY. This module will function either as a transmitter or as a receiver. The module will distribute the

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE012

RF reference signal, provide low-jitter clocks and trigger signals as well as beam-synchronous data such as a timestamp, revolution counters, beam modes, bunch pattern and bunch current via a dedicated optical fibre network. Storage ring, booster and linac will each be equipped with central timing system components, synchronized across the overall facility. Beamline experiments can make use of the same MTCA-based timing system hardware to exploit all accelerator-provided timing system information.

Dependent on the location in the accelerator, the MTCA systems will be equipped with additional AMC, FMC (FPGA Mezzanine Card) and RTM modules for specific read-out, measurement and control tasks, e.g. interface boards to the beam position monitor front-end electronics or feedback controllers as well as ADC boards to acquire measured bunch current pulses or HV-pulses generated by the injection and ejection elements.

All of these modules can be managed remotely via Intelligent Platform Management Interface (IPMI) interfaces and the MCH. Linux drivers with hot-swap support allow for PCIe access of the various AMC and RTM modules from applications running on the CPU AMC.

Likewise, the hardware interfaces for conventional slow-control applications (e.g. magnet power supplies, vacuum system, movable girders) will be compliant with industrial process control standards (e.g. OPC UA, Profibus, EtherCAT, CANopen, USB, RS232/485) preferably providing a well-established and widely-used industrial API.

Special emphasis will be put on the OPC UA interface technology [7]. All power converters for magnets as well as power supplies for getter pumps of the vacuum system will implement an OPC UA server. DOOCS provides a generic bridge server, which seamlessly integrates OPC UA devices into the accelerator control system.

Front-end hardware systems based on the Beckhoff controller technology (EtherCAT/OPC UA) [8] will also be used in many cases, e.g. to control the motors of the insertion devices or of the movable girders supporting all accelerator components. In addition, classical PLC system have to be interfaced. In both cases, generic bridge servers to the accelerator control system are available.

Data Acquisition and Data Archiving

The systems and tools for data acquisition and data archiving have to process time series data as well as snapshot data.

Time series data include data from fast as well as slow data streams. Examples of fast data are data synchronous to the beam orbital frequency (130 kHz), such as that used by low-level RF control algorithms, or magnetic current readback values (41 kHz) used to monitor and analyse power converter performance. Slow data includes data that is updated asynchronously at less than 100 Hz, such as multiturn orbit data or measured vacuum pressures.

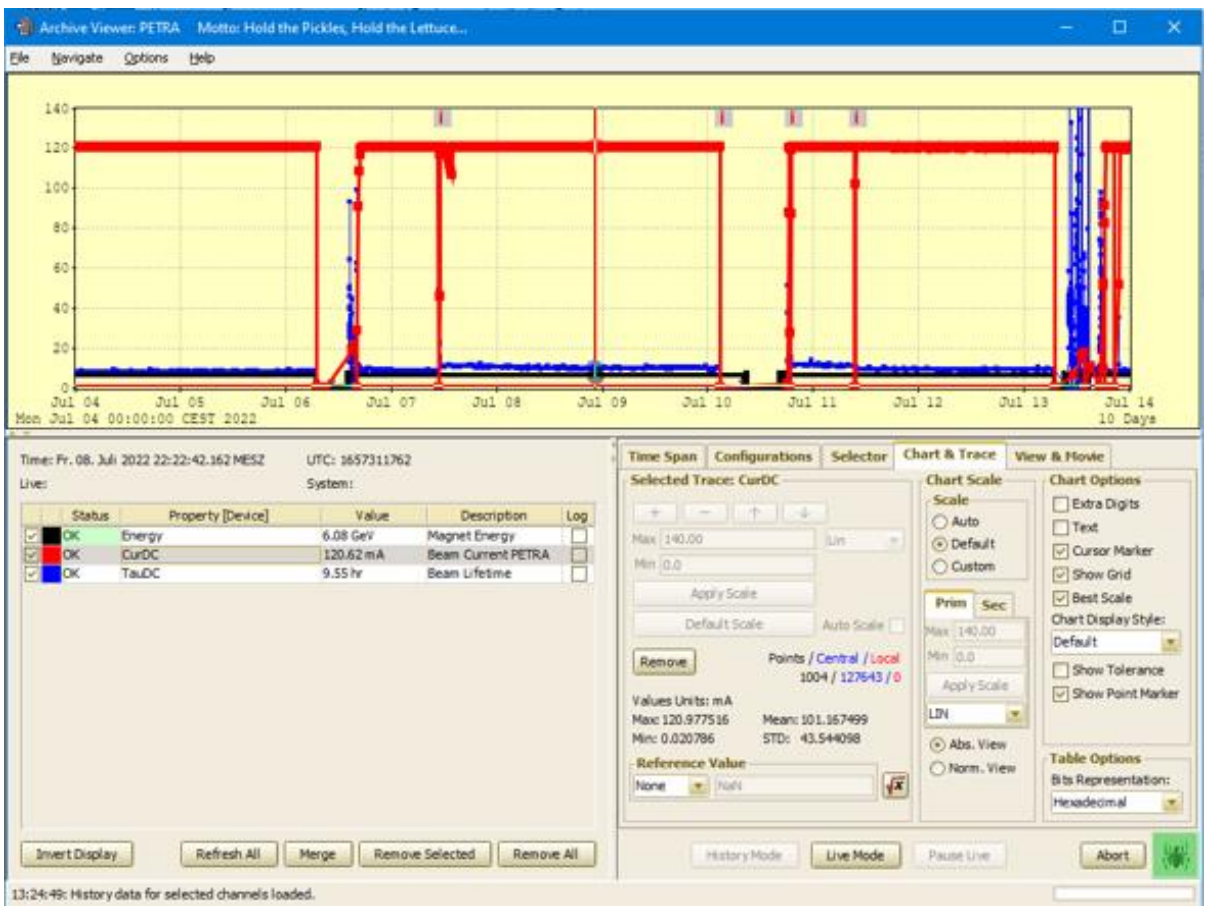


Figure 2: Archive data viewer (PETRAIII).

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE012

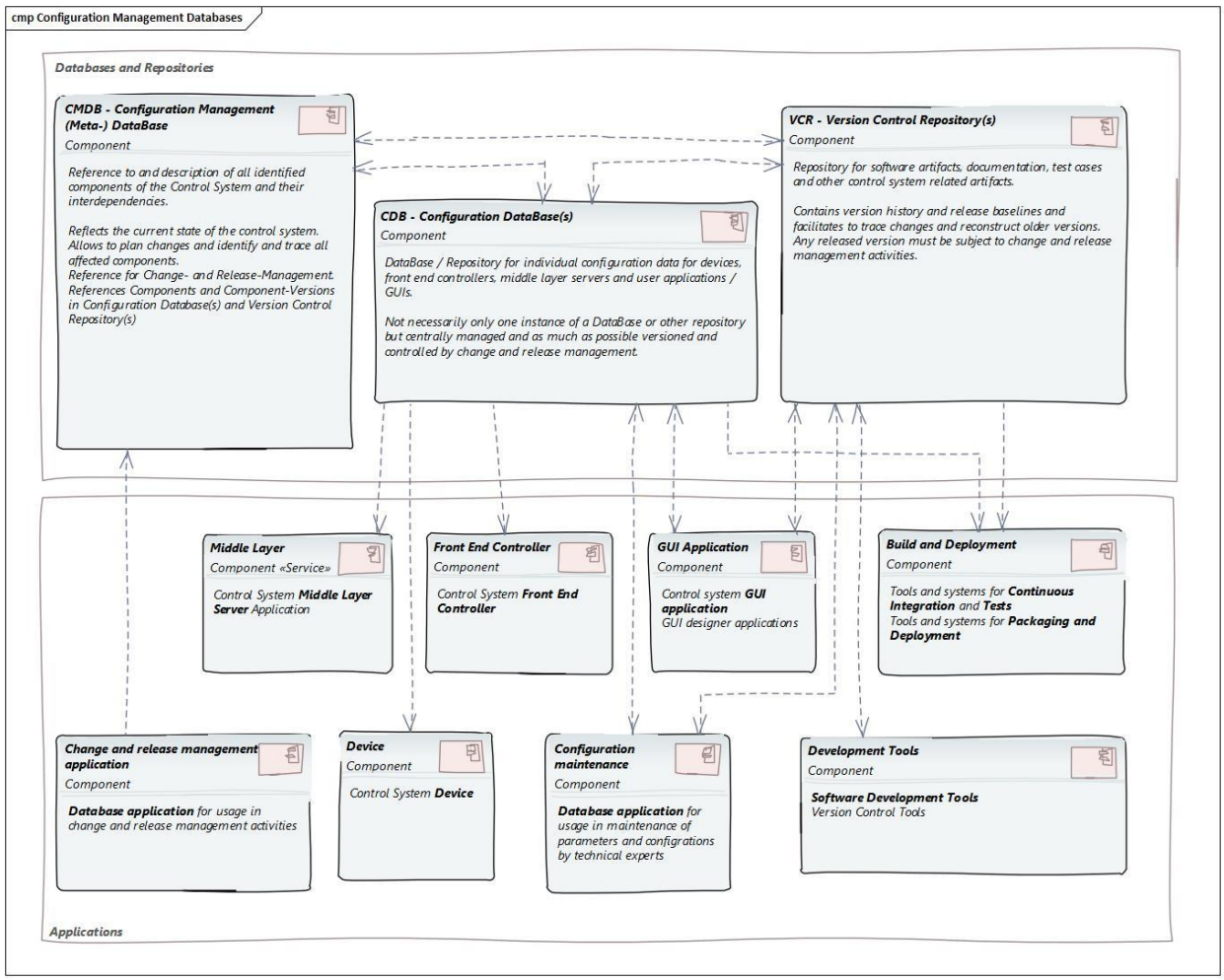


Figure 3: Configuration management system (design sketch).

Generally, the data from the fast data stream is processed by the front-end device electronics and must be aggregated (e.g., averaged) before being made available to other server processes via the control network. Both the aggregated data from the fast data stream and the data from the slow data stream can optionally be stored locally for a limited period of time and are received by a central archiving service, where multiple filtering algorithms can be applied to reduce the amount of data to be stored. Various databases that are particularly suitable for time series data such as TimescaleDB [9] or InfluxDB [10] are currently evaluated and benchmarked.

Only in special cases sequences of fast raw data of selected devices are transmitted via UDP multicast through the control system and will be processed and stored by the so-called DOOCS DAQ system. This system was especially developed for the acquisition and synchronization of pulsed data, such as those generated by the EuXFEL accelerator, and is currently being revised to handle higher data rates.

Snapshot data are stored in a multi-purpose relational database. They are triggered either by a value change (e.g. in case of a device error), by a specific event (e.g. at beam injection), or by an operator request (e.g. while performing a study).

For both cases, versatile visualization (Fig. 2) and analysis tools have to be provided. Particular emphasis will be placed on the capability to support data science applications operated in either online (e.g. learning feedbacks, learning tuning procedures) or offline (e.g. failure prediction, predictive maintenance forecast) mode.

Configuration Management System

The control system of PETRA IV will consist of a large number of software artefacts. In contrast to PETRA III, the number of hardware components is also considerably larger. A comprehensive system for managing software and hardware components (Fig. 3) in all phases of the PETRA IV lifecycle will be established, e.g., by providing repositories and configuration databases, dedicated management applications, and clearly defined workflows and processes for changes and releases.

High-Level Control Applications

A team of controls experts and accelerator physicists has been already established to interface specific needs of beam commissioning and operations and implement corresponding tools and applications.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE012

The well proven MATLAB Middle Layer library suite [11] supplemented by procedures developed for PETRA III operation will be adapted for further use at PETRA IV. In addition, novel control concepts based on advanced machine learning algorithms are being developed and tested at PETRA III.

Similar to the so-called Virtual European XFEL Accelerator [12] a Virtual PETRA Accelerator infrastructure is being set-up. It will be used to test new concepts, enhancements or just modified and improved applications before they will be put into the field which can potentially save significantly commissioning and machine studies time.

ACKNOWLEDGEMENTS

The authors thank the PETRA IV project team at DESY for constructive discussions and the Helmholtz Association of German Research Centers and the Federal Ministry for Education and Research for supporting the preparatory phase for the future project PETRA IV.

REFERENCES

[1] PETRA IV Conceptual Design Report, https://bib-pubdb1.desy.de/record/426140/files/DESY-PETRAIV-Conceptual-Design-Report.pdf

[2] DOOCS, https://doocs.desy.de

[3] JDDD, https://jddd.desy.de

[4] React, https://reactjs.org

[5] D3.js, https://d3js.org

[6] MTCA.4 Standard, https://www.picmg.org/open-standards/microtca

[7] OPC Foundation, https://opcfoundation.org

[8] Beckhoff, https://www.beckhoff.com

[9] TimescaleDB, https://www.timescale.com

[10] InfluxDB, https://www.influxdata.com

[11] MATLAB Middle Layer, https://github.com/atcol-lab/MML

[12] R. Kammering *et al.*, "The Virtual European XFEL Accelerator", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 578-580. doi:10.18429/JACoW-ICALEPCS2015-TUD3O04

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE013

STRATEGY FOR MODERNIZING A 40-YEAR-OLD ACCELATOR CONTROL SYSTEM*

B. Harrison, Fermi National Accelerator Laboratory, Batavia, USA

Abstract

Modernizing the Fermilab accelerator control system is essential to future operations of the laboratory's accelerator complex. The existing control system has evolved over four decades and uses hardware that is no longer available and software that uses obsolete frameworks. The Accelerator Controls Operations Research Network (ACORN) Project will modernize the control system and replace end-of-life power supplies to enable future accelerator complex operations with megawatt particle beams. The project team is evaluating three design concepts, and the future deployment of artificial intelligence capabilities for accelerator operations is an important consideration. An overview of the ACORN Project will be presented, including R&D used for evaluating the conceptual designs in the context of requirements for future accelerator operations.

ACORN

Accelerator Controls Operations Research Network (ACORN) will modernize the accelerator control system and replace end-of-life accelerator power supplies to enable future operations of the Fermilab Accelerator Complex with megawatt particle beams. ACORN is a DOE 0413.3B project with an estimated project cost from $100M to $140M and a lifetime of 8 to 10 years. As an upgrade of the accelerator complex, the ACORN team is analyzing risk associated with current accelerator operations and is developing requirements for modernization.

THE CURRENT CONTROL SYSTEM

Fermilab's accelerator control system has enabled major scientific discoveries such as the top quark, Tevatron collider program, g-2 anomalous muon magnetic moment measurement, etc., all without taking beam down for major upgrades. All upgrades have been incremental without interrupting the beam delivery program.

ACORN is Fermilab's first opportunity to take a large-scale and deliberate approach to modernizing the control system by "standing on the shoulders of giants": building on the success of the current system, upgrading to modern architectures, hardware, user interfaces, software, development processes, documentation, and integrating with modern toolkits like EPICS.

ACORN is taking a requirements-centric approach, first by gathering functional requirements of the current system from interviews with the division; we're making sure we don't lose functionality when we upgrade.

Using requirements as a guide, ACORN is building an aggressive R&D plan to inform our conceptual and technical design process. We will work with AD experts to investigate and prototype ambitious new systems that take advantage of cutting-edge technology, with a focus on standardization and compatibility across the entire division (including with PIP-II).

Requirements and R&D will enable exotic new uses of the control system, such as applied robotics and AI/ML.

We're working with Idaho National Lab (INL) with their human factors team to develop a set of user interface and user experience design standards and frameworks to improve accessibility.

REQUIREMENTS AND RISKS

ACORN in coordination with the accelerator division is performing risk analysis for accelerator operations risks. As a part of that risk analysis, the risks will be categorized based on criticality and cost. Those categories will be used to rank the risks as a priority list to be addressed by the project.

Using the risk ranking, ACORN will define project scope in conjunction with operational and other project risks that may be discovered.

We know that we have more work than our funding can support. The risk ranking allows us to prioritize and include the most important work in the project scope.

For the systems that need modernization, we will develop functional requirements for the existing accelerator control system to help ensure there's no loss of functionality in the modernization process.

In addition to functional requirements of the existing system, we will identify new requirements needed to implement new capabilities.

RESEARCH AND DEVELOPMENT

ACORN is beginning the R&D process to establish knowledge required to define our conceptual design.

Some topics we know we will investigate include:

- Explore new data acquisition capabilities
- Support for AI/ML for accelerator operations
- Evaluation of 5G technology
- Support for robotics
- Evaluation of Experimental Physics and Industrial Control System (EPICS)

Data Acquisition

Our requirements strongly suggest a centralized architecture and we have begun exploring the implications of that choice. In Figure 1 you can see a preliminary block diagram of key components.

We think that a Data Lake structure for our centralized storage will allow us the flexibility to add storage systems as we implement new tools. Key features of the Data Lake we need to test include:

- Measure expected data flows from different types of front-ends

* FERMILAB-TM-2783-AD

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE013

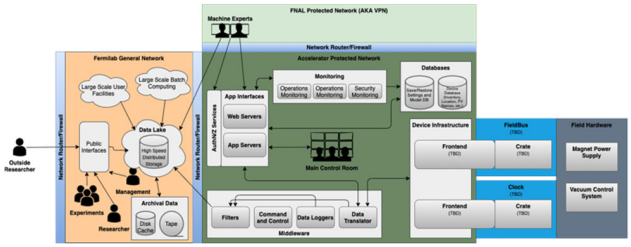


Figure 1: Centralized architecture.

- Create a testbed Data Lake
  - Measure total data volumes
  - Test filters to select and store relevant and interesting data
  - Measure commit latencies for writing data
  - Measure latencies for reading data
- Determine if a Data Lake satisfies accelerator control system requirements

Support for AI/ML for Accelerator Operations

The Accelerator Division mission for AI/ML is to enable safe and reliable improvements to accelerator operations by applying modern artificial intelligence and machine learning (AI/ML) techniques.

- Investigate machine learning operations (MLOps) frameworks to develop, deploy, and monitor AI/ML applications for accelerator operations.
- Design MLOps processes to ensure verification, validation, assurance, and trust for data collection and model building, deployment, and monitoring

Evaluation of 5G technology

The Department of Energy (DOE) tasked the ACORN project with evaluating the uses of 5G wireless technologies in the operator of the accelerator systems.

We are considering the complexities of our use cases and we are planning R&D for the following topics:

- Multiple bands on a single radiating coax antenna
- Silicon equipment lifetime in a radiation environment
- Interference with accelerator equipment
- Signal propagation in accelerator tunnels

Support for Robotics

Development and proliferation of robotics at Fermilab will bring novel technical challenges to the future accelerator control system. Technical challenges include the following:

- Low-latency feedback and control between robot and operator to minimize errors.
- High-resolution and frame-rate video streaming for navigation feedback and reconnaissance.
- Transmitting and logging data from a diverse array of sensors such as temperature, humidity, photography, thermal imaging, and radiation mapping.
- Data storage and computation for training robots to autonomously navigate and manipulate tooling for maintenance and installation tasks.

Human Factors Design Principals

The ACORN team has been working with INL Human Factors team on interviewing operators and developing a guide for intuitive user interfaces.

We plan to build our framework around this guide to give a sane set of defaults to our software developers.

CONCLUSION

ACORN is taking a diligent approach to project management using risk analysis and requirements gathering that will enable us to fulfill the requirements of the accelerator control system users effectively and efficiently.

ACKNOWLEDGEMENTS

Thank you to the ACORN team for all their various contributions to bring this document together.

APPENDIX A
THE ACORN TEAM†

Maria Acosta, Lila Anderson, Eileen Crowley, Dr. Erik Gottschalk, Beau Harrison, Dr. Tia Michele, Anthony Tiradani and Adam Watts
Fermi National Accelerator Laboratory, Batavia, IL, USA

Dr. Katya Le Blanc, Rachael Hill and Zachary Spielman
Idaho National Laboratory, Idaho Falls, ID, USA

† As of Oct 2023

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE022

APPLICATIONS OF TIMING READ-BACK SYSTEM IN J-PARC MAIN RING

M. Yang†, N. Kamikubota, K.C. Sato, N. Kikuzawa, J-PARC Center, KEK&JAEA, Japan
Y. Tajima, Kanto Information Service, Tsuchiura, Japan

Abstract

Japan Proton Accelerator Research Complex (J-PARC) timing system has been in operation since 2006. Over the past 16 years, there were trigger-failure events, and some of which seriously affected the operation of J-PARC accelerator, especially at Main Ring. To troubleshoot the source of such events more quickly, we decided to develop a timing read-back system to read back distributed timing signals at the device side. A PLC-type triggered scaler module was developed as a key of the system. It can count number of pulses in an accelerator cycle and store the counts in a momentary array. Using the module, customized read-back applications for various timing-related signals have been developed: (a) read-back a 25-Hz trigger clock, (b) read-back a pulsed bending magnet trigger, (c) read-back a magnet power-supply trigger. These applications were implemented successfully in J-PARC Main Ring, and demonstrated as countermeasure against past trigger-failure events. More details will be given in the paper.

INTRODUCTION

J-PARC (Japan Proton Accelerator Research Complex) is a high-intensity proton accelerator complex. It consists of three accelerators: a 400-MeV Linac (LI), a 3-GeV Rapid Cycling Synchrotron (RCS), and a 30-GeV slow cycling Main Ring Synchrotron (MR) [1, 2]. Since the initial beam in 2006, J-PARC has been improving beam power. Concerning MR, recent beam power is about 500-kW (50-kW) by fast (slow) extraction, respectively [3].

There are two time cycles used in J-PARC. A rapid cycle, 25 Hz is used at LI and RCS, and a slow cycle is used at MR. When MR delivers proton beams to Neutrino (NU) and Hadron (HD) Facilities, 2.48-s (fast extraction mode (FX)) and 5.20-s (slow extraction mode (SX)) cycles are used, respectively. Because the slow cycle determines the overall time behaviour of the accelerators, it is also called a "machine cycle."

The control system for J-PARC accelerators was developed using the Experimental Physics and Industrial Control System (EPICS) framework [4]. In addition, a dedicated timing system has been developed and operated since 2006 [5, 6]. It consists of one VME transmitter module and approximately 200 VME receiver modules. Event codes, which correspond to the information on beam destinations and on beam parameters, are distributed from the transmitter module to all the receiver modules. A fiber-optic cable network is used for event-code distribution using several optical-to-electrical (O/E) or electrical-to-optical (E/O) modules. According to the received event code, each receiver module generates eight independent delayed-trigger signals.

Since the first beam use began in 2006, the J-PARC timing system has contributed to a stable operation of the accelerators [6]. Nevertheless, some timing trigger-failure events have occurred during beam operation. During each recovery process against a failure, it was often difficult to find a definite module among many modules suspected. Such experiences have prompted us to develop a new module, triggered scaler module, which can read back signals generated by the J-PARC timing system [7, 8].

In this paper, three trigger-failure events occurred in J-PARC MR are introduced briefly. The applications of timing read-back system are described in details, followed by a future plan of new power-supply with embedded timing modules.

PAST TRIGGER-FAILURE EVENTS

There have been some unexpected trigger-failure events in the past 16 years [8, 9]. Herein, three events are briefly given.

- In November to December 2016, the accelerator operation was suspended several times a day, because of the faults of a beam diagnostic system. The investigation showed that an O/E module used for 25-Hz trigger clock produced irregular signals. After replacing the module, the problem was solved.
- In January 2018, a delayed trigger which excites the pulsed bending magnet was stopped. Thereby, few miss-controlled beams went to an undesirable destination, Material and Life Science Experimental Facility (MLF). Soon we found that a fuse in a trigger-fanout module was broken. After replacing the module, the problem was solved.
- In November 2015, a bad quality beam appeared during stable beam delivery to HD. Such beams appeared a few times per month [7]. After about six months troubleshooting, we finally found that a timing receiver module for one of the MR steering magnets showed momentary errors by external common-mode noises. The problem was solved by adding ferrite cores to metal cables.

The first two events showed that a single failure of one module causes a critical problem in an accelerator. It showed no alert from neither the timing system nor the control system. Thus, it was unable to find the problems remotely. The third event was a very low-rate error, a few

† yangmin@post.kek.jp

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE022

times per month. It was very difficult to search for the trouble source.

These events led us to realize that it was necessary to develop a new system. The system is expected to read back timing signals at suspicious points, and raises an alert when a fault is detected.

TIMING READ-BACK SYSTEM

Introduction

The timing read-back system has been developed as an independent system from the existing timing system. It observes and confirms the delayed-trigger signals, which are provided from the timing system. A unique device, triggered scaler module, is used in the read-back system.

Triggered Scaler Module

Triggered scaler (hereafter TS) module, designed by J-PARC control group, is the key device for developing the timing read-back system. It is a scaler to count number of pulses with respect to the machine cycle. The module stores observed counts in a momentary array. One TS module has four independent input channels. The details of the hardware design, the working principle, and the performance can be found in [8-10].

In 2020, the firmware of the TS was updated [9, 10]. The customized applications were developed based on the updated TS module.

Prototype System

Based on early measurement of the TS module in 2018 [7], a prototype timing read-back system was developed in 2020. The hardware of the prototype system is shown in Fig. 1. It consists of a CPU module, a TS module, and a power supply module. All of them are standard Yokogawa PLC modules. Linux and EPICS are running on the CPU module.

In June, 2020, the prototype system was tested for three types of trigger-failure events, using a dummy injection-kicker signal. As shown in Fig. 2, the system successfully detected the dummy signal and identified all kinds of failures. The details of the system development and discussion on possible types of signal failures are given in elsewhere [8].

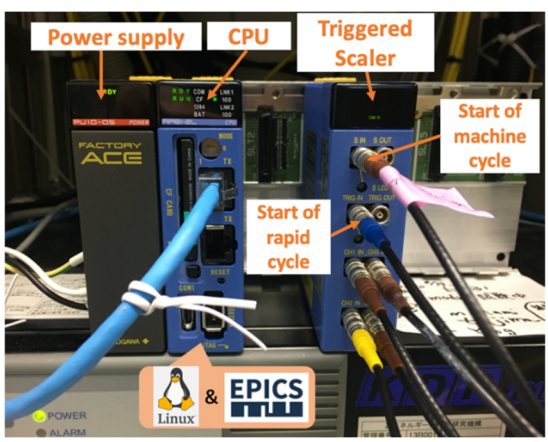


Figure 1: Hardware setup of the prototype system.

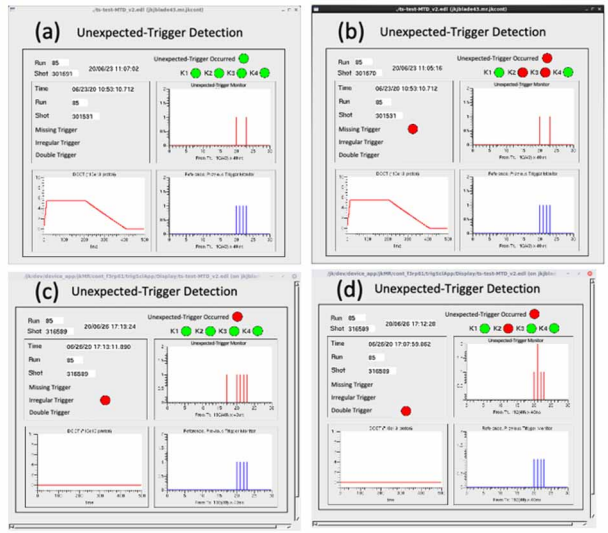


Figure 2: Test results of the prototype system using a dummy injection-kicker signal. Red means an observed signal, blue means a reference signal. Screenshots for (a) normal event with the saved information of the last fault event, (b) a missing-trigger event, (c) an irregular-trigger event, and (d) a double-trigger event.

CUSTOMIZED APPLICATIONS OF TIMING READ-BACK SYSTEM

After the prototype system, some customized applications of timing read-back system were developed toward stable operation of J-PARC MR [9, 10]. Here shows three applications.

Read-Back System of 25-Hz Trigger Clock

The read-back system of 25-Hz trigger clock was developed and tested at one power-supply building of MR in 2021 [10]. Afterwards, the system was upgraded to cover three power-supply buildings. Figure 3 shows the GUI for one of three buildings. Since June, 2022, the status indications of all the three buildings are shown together with those of magnet power supply trigger (Fig. 5).

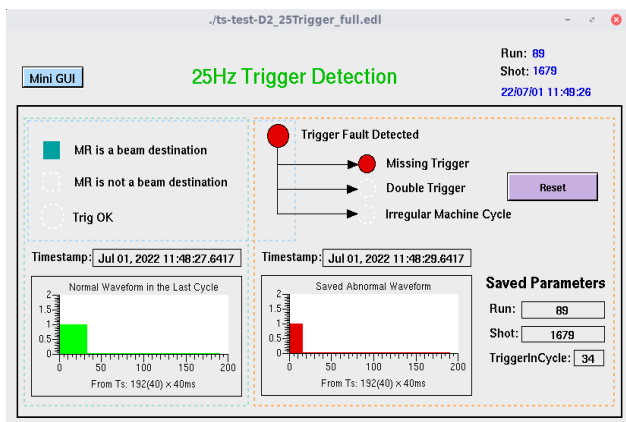


Figure 3: The GUI screenshot of 25-Hz trigger clock for one of power-supply buildings. The indicated error is caused by a simulated missing-trigger signal. Red means an observed signal, green means a reference signal.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE022

Read-Back System of Pulsed Bending Magnet Trigger

The read-back system of pulsed bending magnet trigger was developed in 2021 [9]. Since then, it has been operational during the beam operation of J-PARC MR. Fig. 4 shows the screenshot of the GUI.

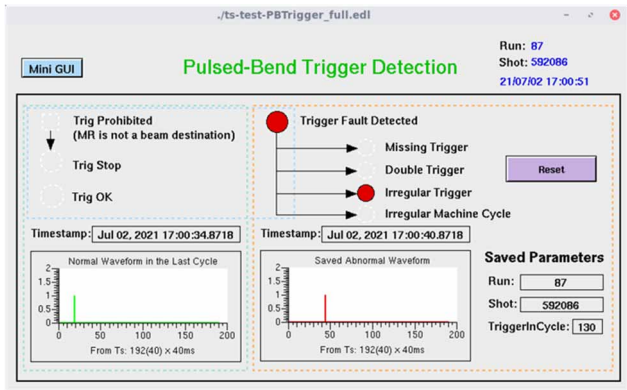


Figure 4: The GUI screenshot of pulsed bending magnet trigger with a simulated irregular-trigger event.

Read-Back System of Magnet Power-Supply Trigger

The read-back system of magnet power-supply trigger was developed in 2022. The hardware consists of three PLC-based setups to cover three power-supply buildings. Each of the setups supervises both 25-Hz trigger clock and magnet power-supply trigger. The system has been operated during the J-PARC beam operation in June and July, 2022. Fig. 5 shows the main GUI.

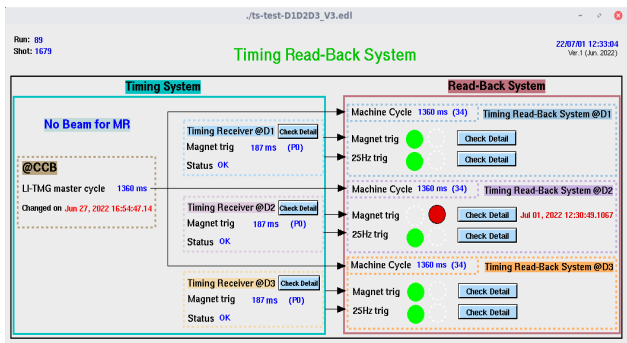


Figure 5: The GUI screenshot of 25-Hz trigger clock and magnet power-supply trigger. The "Timing System" part shows timing system related information. The "Read-Back System" part shows the observed signal status of three power-supply buildings. The indicated error is caused by a simulated missing-trigger signal.

Discussion

After June, 2022, all the three applications of timing read-back system have been operational. They are actually countermeasures against past trigger-failure events given in the above section. With the applications, it becomes possible to find a trigger-failure event remotely. So far, no trigger-failure event was observed.

READ-BACK SYSTEM FOR OTHER SIGNAL

Besides three read-back applications, a read-back application for other signals was also developed, an LLRF pattern monitoring system. In 2018, it was developed to visualize LLRF signals for J-PARC MR [7], and implemented for beam operation in 2021. Figure 6 shows the observed LLRF patterns in different MR energies during May and June 2021.

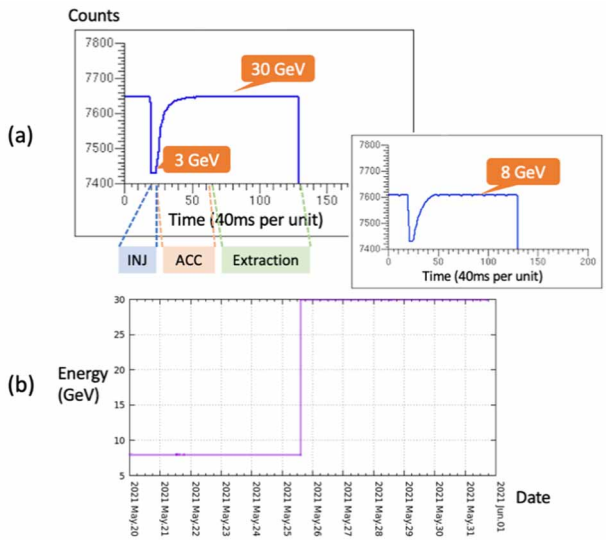


Figure 6: (a) The frequency shift of the LLRF signal is observed by the read-back system. In every machine cycle, 3-GeV protons are injected into MR, then accelerated up to 30-GeV or 8-GeV. (b) 10-day history of the MR energy, which is deduced from the observed LLRF patterns. The energy was changed from 8-GeV to 30-GeV on May 25, 2022.

FUTURE PLAN FOR PULSED BENDING MAGNET

Experienced past trigger-failure events showed that the aging of timing modules has been major sources of failures. To proceed scheduled replacement of old modules with new ones, a next-generation timing system for J-PARC has been discussed [11]. The new timing receiver for MR was developed as a PLC-type module. It is still in test phase and not in use so far.

Besides, a new pulsed bending magnet power supply is under construction along the MR's 1.3-MW power-upgrade program [12]. It will be installed to J-PARC in 2023. The first PLC-type receiver is planned to be used.

Figure 7 shows the picture of the read-back system with a new PLC-type receiver. Both a receiver module and a TS module are mounted in the same PLC-based EPICS setup, which will be embedded in the frame of the new power supply. In addition, a digitizer module will supervise the output current of the power supply.

As shown in Fig. 8, compared with the current system, there will be no O/E and E/O module. We expect that this scheme will reduce the failure rate drastically.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE022

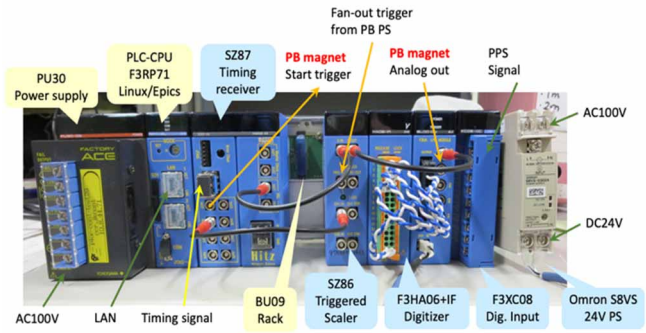


Figure 7: The picture of the read-back system with a new PLC-type timing receiver.

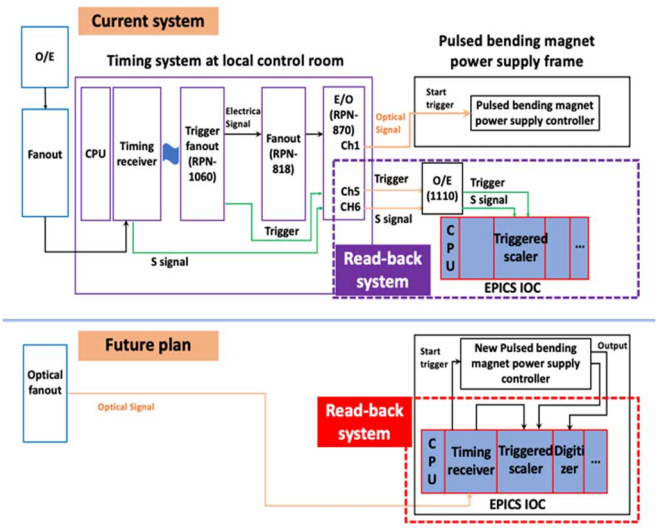


Figure 8: Comparison of current and future timing read-back systems for pulsed bending magnet trigger.

CONCLUSION

Three customized applications of timing read-back system, (a) a read-back system of 25-Hz trigger clock, (b) a read-back system of pulsed bending magnet trigger, and (c) a read-back system of magnet power-supply trigger, have been developed in J-PARC MR. These applications are demonstrated as countermeasure against past trigger-failure events, and are in operation since 2021 and 2022. If the same trigger-failure event as before occurs again, the applications will detect it immediately. It is expected to reduce the time to find the source of failure.

Another read-back application, an LLRF pattern monitoring system, was also developed and started supervising an LLRF signal since 2021. The system is helpful for accelerator operation to confirm the MR energies. It is worth noting that the triggered scaler module can be applied to non-timing field.

A future plan of new power supply for pulsed bending magnet is described. A new PLC-type receiver module and a triggered scaler module will be embedded in the frame of the power supply.

We expect that these works will contribute more reliable operation of J-PARC MR.

ACKNOWLEDGEMENTS

We express our best thanks to Dr. S. Yamada, KIS members, and other J-PARC MR staff members for their encouragement and suggestions during system development.

Special thanks go to Dr. M. Shirakata and Dr. J. Takano for various tests with the pulsed bending power supply.

REFERENCES

[1] J-PARC website: http://j-parc.jp/index-e.html

[2] S. Nagamiya, "Introduction to J-PARC", *Prog. Theor. Exp. Phys*., vol. 2012, p. 02B001, 2012. doi: 10.1093/ptep/pts025

[3] S. Igarashi *et al.*, "Challenging to Higher Beam Power in J-PARC: Achieved Performance and Future Prospects", in *Proc. 10th Int. Particle Accelerator Conf. (IPAC'19)*, Melbourne, Australia, May 2019, paper MOYPLM1, pp. 6-11.

[4] N. Kamikubota *et al.*, "J-PARC control toward future reliable operation", in *Proc. ICALEPCS'11*, Grenoble, France, 2011, pp. 378-381.

[5] F. Tamura *et al.*, "J-PARC timing system", in *Proc. ICALEPCS2003*, Gyeongju, Korea, 2003, pp. 247-249.

[6] N. Kamikubota *et al.*, "Operation status of J-PARC timing system and future plan", in *Proc. ICALEPCS'15*, Melbourne, Australia, 2015, pp. 988-991.

[7] N. Kamikubota *et al.*, "Development of triggered scaler to detect miss-trigger", in *Proc. PCaPAC'18*, Hsinchu, Taiwan, 2018, pp. 213-215.

[8] M. Yang *et al.*, "Applications of triggered scaler module for accelerator timing", in *Proc. 22nd Virtual IEEE Real Time Conference*, 2020. doi:10.48550/arXiv.2010.14716

[9] M. Yang, "Development of Timing Read-Back System toward Stable Accelerator Operation", Ph.D. thesis, The Graduate University for Advanced Studies, SOKENDAI, Japan, 2021.

[10] M. Yang *et al.*, "Development of Timing Read-Back System for stable Operation of J-PARC", in *Proc. 18th Conf. on Acc. and Large Exp. Physics Control Systems*, Shanghai, China, 2021, pp. 927-930.

[11] F. Tamura *et al.*, "Development of next-generation timing system for the Japan Proton Accelerator Research Complex", IEEE Transactions on Nuclear Science, 2021. doi:10.1109/TNS.2021.3083791

[12] S. Igarashi *et al.*, "Accelerator design for 1.3-MW beam power operation of the J-PARC Main Ring", *Prog. Theor. Exp. Phys,* vol. 2021, no. 3, March 2021. doi:10.1093/ptep/ptab011

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE023

AVOIDING AND MANAGING PITFALLS OF CONTROL SYSTEM PROJECTS

A. Jesenko*, G. Pajor, Cosylab d.d., Ljubljana, Slovenia

Abstract

Cosylab is the leading provider of software solutions for the world's most complex, precise, and advanced systems. We also provide software products and services to the largest medical device manufacturers and cancer centres worldwide. In our 20-year history, Cosylab has worked on hundreds of multi-year projects worldwide.

Software projects run the highest risk of cost and schedule overruns. On average, large IT projects run 45 percent over budget and 7 percent over time.

There are some common pitfalls when defining and executing projects in the Control Systems field. This article presents some concepts on how to address them, such as: having a clear definition of a project's scope, budget, and timeline; doing project risk management; having a clear division of responsibilities; and having a project sponsor from the management.

INTRODUCTION

Control system development is an engineering discipline like many the others, often with a quite complex life cycle. Jumping right into prototyping and coding can bring some initial results quickly but is very prone to a myriad of risks and undesired results, such as an unclear scope, a lack of acceptance criteria and vaguely defined responsibilities. This can consequently manifest in never-ending development, an unnecessary multiplication of effort and non-existent or sub-par documentation.

Control systems are an engineering discipline like all the others, but with an even more complicated cycle:

1. Write specifications
2. Architecutre
3. Design
4. Prototyping – probably the only fun part
5. Test procedures
6. Implementation (coding) – the only software part
7. Documentation
8. Testing
9. Debuging
10. Acceptance

Each control system project has the following entities

- Consumer: the entity that is ordering or is otherwise interested in getting parts of (or the whole) control system.
- Supplier: the entity that is supplying parts of (or the whole) control system to the consumer.

A control group can have either of these roles; it can be the consumer ordering the control system project from another entity and it can also be the supplier supplying the control system to their own or another institution. One part of the control group can also assume one role, and another part of the same group can assume the other role. In all these cases the basic concepts described in this article remain the same.

PROJECT PLANNING

Quite intuitively, project planning is a crucial process for any project. The better you define the project and how it will be closed in the beginning, the less unfinished scope will remain in the end. This can result in a lower total cost of ownership. Remember that there will be some work after the project is complete and this work should also be planned for. In any case, don't count on defining the project scope while the project is ongoing (see also "What about Agile?").

The basic elements that define a project are:

1. the scope (what will be done, and who will do it),
2. the budget (how much will it cost),
3. the risks (what can go wrong).

The first steps in creating a project plan are to define the scope of the project, and to find out the time and budget constraints. After that, the scope is broken down into a "work breakdown structure" (WBS), and the risk management process is started.

Project Scope

The project's scope is usually documented into a statement of work (SoW). This is an important document with details about:

- What was agreed would be done, and by whom.
- What will not be done in the scope of the project.
- What are the deliverables.
- How acceptance of deliverables will be done.
- What assumptions were made.

Project Timeline and Budget

A rule of thumb to estimate the budget for an accelerator control system is to take 10 - 15 % of the total budget. To start planning a more detailed timeline and budget, work should first be broken down into a work breakdown structure (WBS). An effort estimation should then be done for each task in the WBS. This is a good way to gain knowledge about how much effort from people of each profile you will need for the project. Note that the effort estimations will only be accurate if they are done by persons experienced doing them.

Once we have a WBS done, we need to know when we will be ready to start each of the tasks. Now it's useful to create a critical path. By looking at a critical path, we can tell prerequisites and a timeline for each of the tasks and the project in general.

Then we can set some milestones for the project. It's a good idea to plan monthly milestones, in which the team

* anze.jesenko@cosylab.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE023

delivers or shows something. This way it's easier for both the team and the manager to objectively see how well the project is doing, whether there are some issues that must be resolved …

Project Risk Management

While project risk management is widely recognized as a beneficial activity, it's also widely forgotten about during actual project management. Even when it's being done, many times a risk assessment is done at the beginning of the project and then later forgotten about during the project. For a project to be a success, it is necessary to monitor the risks and act on the spot. To be able to do this, the risk manager should document measures and procedures for each identified risk:

- A measure that reduces the chance of the risk materializing.
- A procedure to perform if the risk materializes, a "missed approach procedure" [1] in aviation terms. This should be done so that you are not required to think in panic.

What About Agile?

Agile software development is probably a good idea, especially for internal control groups. But it is worth understanding that with large control systems projects, there are many things that must be agreed in the beginning of the project, or even before that. Otherwise, some of the participants of the project will not be able to work and deliver in time, and this will cause delays. But if critical specifications and interfaces are agreed up front, then doing the software development the Agile way is ok.

In short, for internal control groups Agile is a good idea and we recommend it, but not as a replacement for the definition of the project.

PROJECT EXECUTION

Rather than giving lengthy instructions on how to execute a project, we are focusing in this article on a few ways how a project manager can avoid the most common pitfalls of control system projects.

Change Control

Uncontrolled change is probably the most common reason of failure of control system projects. When we talk about change, we talk about change in any dimension of the project definition (scope, timeline, or budget). While some changes in scope are necessary in most projects, it is worth understanding that an increase in either scope or timeline results in an increased cost of the project. To protect the success of the project, any change must be accepted in a controlled way.

The Agile methodology inherently handles change by planning work in smaller packages. When not using an Agile methodology, a change control procedure must be established [2].

A summary of a basic change control procedure is:

1. A change is identified and reported by anyone involved with the project. The importance of the change is estimated.
2. Impact assessment is done. What is the impact on the budget and timeline? Are there any risks involved?
3. Review and approval of the change is done by a decision-making body. Changes with a high impact and low importance are usually rejected, and changes with a low impact and high importance are usually approved.
4. The project specifications are formally updated.
5. The change is implemented.

Single Source of Truth

It is important that a single source of truth is established for each project. This is some location (like a cloud share) where anyone involved in the project can access and add their latest versions of specifications, meeting notes, source code …

It does not matter much what exactly the implementation of this system is (like a Git revisioning system for source code, a document management system for documentation). But it does matter that every participant uses it and that it is clear to everyone that the truth is only what is stored in the system.

Regular Communication

Regular communication is important to build and keep good relationships, to be sure that everyone understands the project the same way, to provide a place where everyone can report issues and voice concerns …

Once the project has started regular communication paths have to be established. All stakeholders must be considered (team, management, outsourcers …), and the type (sync meetings, mails …) and frequency of communication must be agreed. During sync meetings, it is a good practice to note down meeting minutes and action items and store them in the single source of truth. Action items should be a regular part of the agenda. If an action item is not done on time, it should be discussed what the reasons for this. Participants of the meeting can offer insight and help with any issues. Issues that cannot be resolved can be escalated.

Escalation Path

Sometimes an issue cannot be resolved within the project team. To keep relationships between team members intact, it's a good idea to have sponsors from management assigned to each project. A "project steering committee" should be established already when planning the project, and a meeting schedule should be agreed. The steering committee should be composed of the project manager(s) and persons with decision power, e.g., the people responsible for the project resources like personnel and budget. During a steering committee meeting the project manager presents the status of the project and the escalation requests. The committee provides the project manager decisions on the course of action for each escalation.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-WE023

PROJECT CLOSURE

To start closing the project, first revisit the SoW document to check that everything that was planned for was done. Acceptance testing of deliverables should be done whenever possible to check that the goals of the project were achieved. The consumer (entity that is ordering the project) must agree with both the testing procedure and the results. Once tests pass without dispute, the deliverables are accepted.

But project closure isn't just about the completion of acceptance tests. Consider and plan also for how the transfer of knowledge will be done, who will maintain and upgrade the solution, and who will provide support to users and the maintenance team. Take some time to think about the lessons learned and document and discuss them with the team. Look at the initial effort and timeline estimates and compare them to what happened.

And don't forget to buy the team some pizza and beer.

CONCLUSION

Due to control system's specific role, it interfaces almost all other systems. This consequently means that it relies on information and deliverables coming from these systems and for the most part, delays on these prerequisites also cause a delay of the control system project. This is further emphasized by the fact, that the control system is usually one of last things to be finished and can lead to erroneous conclusion that the control system is responsible for the overall delay. To avoid such an unfavourable situation, good planning and adequate communication with all relevant stakeholders is vital – on top of well managed project execution of course.

REFERENCES

[1] Missed approach, https://skybrary.aero/articles/missed-approach

[2] Change control, https://en.wikipedia.org/wiki/Change_control

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

BUILDING CONTROL SYSTEM REMOTELY

M. Lukaszewski*, K. Klys, E9 Controls Limited, London, England

Abstract

Building a control system for a scientific facility is a complex process that requires significant time coordination and the cooperation of hundreds of people. The latest of our control system for the 10 J 100 Hz laser system happened to be built when access to the lab was far more difficult, due to the pandemic: travel was much more complex, and local restrictions at one point prevented a large number of people from being in the lab simultaneously. This paper shows how, despite the demanding working conditions, we built the control system for the 10 J 100 Hz laser. The control system was designed in collaboration with CLF (Central Laser Facility, Science and Technology Facilities Council, UK) and HiLASE (Institute of Physics, Czech Academy of Sciences). The process of building the laser control system was divided into stages and we had to rely on remote working. This article discusses how we adapted each stage to work remotely, what tools we used, how we minimized risks, and what we would have done differently if we had started from scratch.

INTRODUCTION

This paper has been written to share the methods and tools used to build a control system for the 100 Hz 10 J laser system built in collaboration with CLF (Central Laser Facility, Science and Technology Facilities Council, UK) and HiLASE (Institute of Physics, Czech Academy of Sciences). The paper aims to share findings and processes, and evaluate risk awareness.

REMOTE WORK

The main engineering effort started shortly after the first wave of pandemic restrictions, thus making it impossible to work directly in the laboratory. Experience with previous deployments was leveraged, and it was decided that the core integration layer be built remotely without accessing the actual devices. This approach was possible because proper software practices were followed from the beginning, such as: expressive documentation, simulation of various system components, continuous integration during development, automation of repeatable tasks and open communication with stakeholders. All of these aspects will be discussed below.

RISK MITIGATION

Building a scientific system is an arduous and lengthy process. It can be broken down into individual stages, including design, tendering, subsystem construction and equipment integration. The process ends with the implementation of the control system and functional testing of the entire system.

* marcin.lukaszewski@e9controls.com

However, problems with tenders, broken supply chains and other factors affecting the work process, which are not the subject of this article, cause numerous delays that increase the risk of error during integration and negatively affect the developer experience.

To mitigate the risks arising from delays, several strategies have been implemented to react effectively in a rapidly changing environment. It should be noted that a large proportion of the problems that arise during the development of software for controlling equipment are due to a lack of adequate information or late acquisition of such information.

Involvement Therefore, it is essential to participate in the design work, where the parameters of each device are detailed. At this point, each change is "cheap and easy" to make, which provides an opportunity for finding errors regarding device communication and with possible impact on the quality of the integration of devices into the control system.

Regular Contact Also, direct and regular contact with equipment suppliers allows changes in the control software to be tracked, potentially affecting the integration efforts.

Device Grouping Off-the-shelf equipment that was widely available, or was from large suppliers was given the lowest priority. It was assumed that basic integration for these devices would be available in the environment or that the system would be relatively simple to build, given the availability of documentation.

Devices used in the past, or with similar functionality to those already deployed were given medium priority. It was assumed highly likely that software for such devices either already existed in the community, or could be used after modification.

The highest priority was given to new equipment, which was previously unknown and also from suppliers unfamiliar to the laboratory. Equipment from small suppliers was also included in this group. These devices were integrated first, which resulted in the broadest possible time window to develop a working version.

INFRASTRUCTURE

Production Environment – Laboratory

The control system infrastructure in the laboratory consists of five powerful servers that control data collection and devices. The servers are connected via a high-speed Ethernet network to which all devices in the system are connected. The servers run under Linux Ubuntu LTS. The software controlling the system is based on the Experimental Physics and Industrial Control System (EPICS) framework, described later in this article. The lab has other systems with a similar

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

framework, so it was decided to build a system with similar functionality to reduce maintenance efforts.

Staging Environment

By definition, the staging environment should replicate the production environment - the laboratory set-up. A staging site's main purpose is to ensure that all new changes are working as intended before they are deployed on the production servers [1]. With this approach we can limit to a minimum the risk of any software failure in the laboratory environment. At this stage we can detect any memory leaks, excessive CPU or RAM usage or just wrong functioning of the software.

The biggest challenge was to reflect the devices that the laser control system is composed of. To do it, we decided to write custom simulators. Their role was to emulate the connection protocols and behaviour of the specific devices. The emulators allow for IOC functioning verification. Furthermore, the staging environment contains the exact copy of the production monitoring and archiving infrastructure.

Testing Environment

The main task of the test servers is to build and perform test executions of the newly created software components. In terms of the control system software for the 100 Hz laser facility, it was crucial to test all EPICS components - IOCs. In the beginning, two test servers were established using one of the cloud providers. They contained all the necessary software to build EPICS together with support modules and to run/ and test IOCs. The custom tool launches IOCs and sets and reads values of PVs.

To be sure that each IOC has been evaluated successfully, we integrated them with Gitlab Continuous Integration (CI) pipelines. Once a new commit appears in the IOC repository, the GitLab shared runner copies the code, builds it, and runs tests on one of the servers.

Beacuse most of the time, the servers have been waiting for new commits and desire to limit costs, we decided to use Docker images containing test tools and the EPICS environment in GitLab and run new commit changes inside the containers. The main disadvantage of this approach is the total test time. GitLab needs time to download images and all dependencies for each test, so it takes more time to perform all of the actions. Nevertheless, it is not the most important feature for users. It is still acceptable if a test takes from 15 to 30 seconds. Thanks to that approach we do not need to maintain remote servers and we can apply the Infrastructure as Code (IaC) idea, which is the whole configuration in Dockerfile.

EPICS

EPICS is a software framework that provides a set of toolkits destined for designing distributed control systems. It is widely used in large experiments like particle accelerators and telescopes, as well as smaller infrastructures such as lasers or industry systems. EPICS employs Client/Server and Publish/Subscribe mechanisms to handle communications between the various computers. The typical EPICS architecture of the control system is organized in a way that a group of servers - IOCs perform real-time actions (e.g. data acquisitions) and then, using dedicated network protocols, Channel Access (CA), or its newer and faster implementation PV Access (PVA), the data are published to subscribed clients [2, 3].

Compared with other control system packages, EPICS does not model control system devices as objects but rather as data entities that describe a single aspect of the process or device under control, thus the name Process Variable (PV) [2].

EPICS is open-source software and one of its main advantages is the community that constantly improves it and creates helper tools that facilitate control system development and EPICS integration.

DEVICE SIMULATORS

Device simulators are a key element of the testing and staging environments. They allow for IOC tests in the cases of communication protocols and functional tests. This is a crucial aspect in terms of remote IOC development without physical equipment. Based on the documentation we can test the communication and how the IOC should behave in advance.

We chose to use Lewis. This is a Python package that makes it easy to develop complex stateful device simulations [4]. It supports basic industrial communication protocols like TCP and Modbus and it can establish an EPICS server to expose PVs with device parameters. The package does not support custom communication protocols or low-level drivers, but the 100 Hz laser control system is based on devices with standard protocols TCP (e.g. laser rack, delay generator), Modbus (e.g. PLC), or EPICS interface (e.g. Front-End ModBox) [4].

It is worth mentioning that it is not always possible to simulate all the device states'. In most cases, it is more than enough if we can emulate and test the communication between an IOC and a simulator. After that, when there is a possibility to examine an IOC with a real device, it is much easier and faster to implement the IOC parts corresponding to the device states and behavior while having the proper communication ready. That is why there usually is no need to browse and study the documentation to investigate how the device behaves.

In the described 100 Hz laser infrastructure, the Lewis simulators are used to test IOCs in GitLab CI pipelines. Inside the Docker image, the custom tool launches the IOCs with the corresponding Lewis simulators and performs test cases described in the YAML configuration file. In the staging environment, the simulators are launched together with the IOCs to reflect the laboratory setup.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

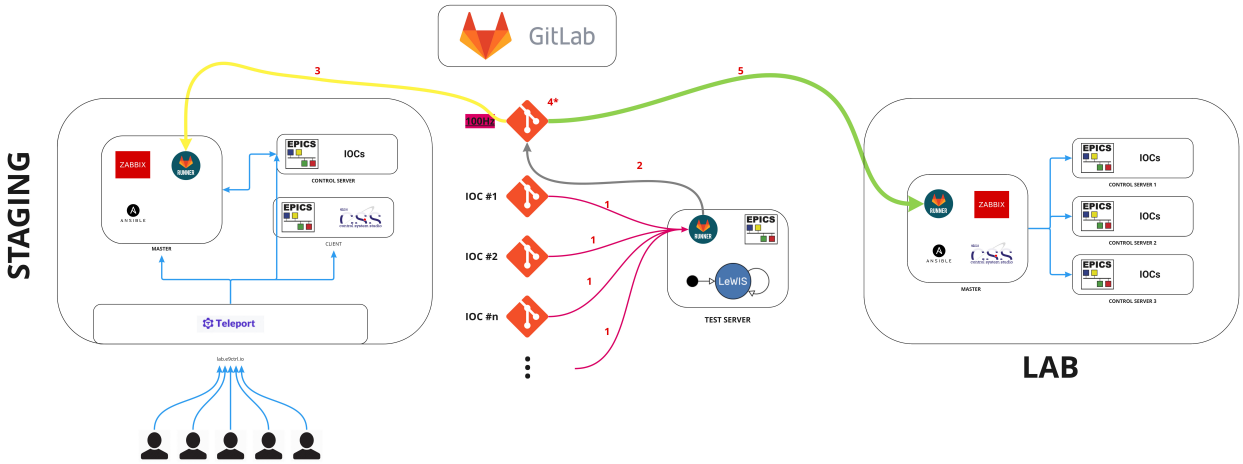


Figure 1: Diagram showing the relationships between the various elements of the system.

AUTOMATION AND CI/CD

Automation is an important aspect of modern IT (information technology) infrastructure. It is the process of creating software and systems to perform repeatable actions and replace or reduce manual intervention. Its main advantage is the acceleration of IT infrastructure delivery by automating processes that previously required a human touch [5].

To limit the risk of mistakes and ease long-term maintenance, we have automated almost all processes: infrastructure build, testing, deployment, and monitoring. This has drastically minimized human engagement, increased system responsiveness, and improved user experience.

Continuous integration is a coding philosophy and set of practices that drives development teams to frequently implement small code changes and check them into a version control repository [6]. We have applied CI practices to IOC testing. Each commit is verified in terms of syntax rules, formatting, and functional tests being performed.

Continuous delivery picks up where continuous integration ends and automates application delivery to selected environments. It is an automated way to push code changes to any environment [7]. We have designed the workflow that tests IOCs and when a new commit appears in the master branch, deploys them to the staging environment. It is presented in the Fig. 1.

Building Infrastructure

To avoid manual server configuration and to facilitate server management, we decided to employ Ansible. For each environment (staging and production) there is a specific playbook with the corresponding list of hosts and variables to set up the proper configuration, i.e. network settings, installation of all required software, and starting of needed processes. The Ansible playbooks are also the infrastructure documentation, and thanks to them, it is easier to follow any configuration modifications.

For tests, the Docker image has been designed. It contains the EPICS framework and all the tools to perform tests inside the container when the GitLab CI pipeline is launched.

Tests

As mentioned in the previous section, the tests are made automatically, after each new commit in the repository. They are performed using GitLab CI pipelines together with custom Docker images, and they are divided into several jobs. The first one checks if all needed files are available - files describing test cases, IOC, and simulator directories. Then, the IOC database files are validated in terms of syntax errors. After that, the IOC is built and installed. The last step is the functional tests. Using the YAML configuration file with the test cases (modify set point value and verify its readback, or check if the PV value has changed over a defined time), the custom called 'ioc-test' performs functional tests by launching IOC together with the simulator and setting and reading PVs characterized in the YAML file.

Deployment

If the IOC tests have been finished successfully, the GitLab CI triggers the deployment job. Its role is to launch the Ansible playbook with the IOC deployment procedure based on the source of the trigger (IOC type). The Ansible role checks if there is a new IOC version in its repository. If yes, it installs it on the staging server together with a new simulator instance. The IOC is automatically added to the monitoring and achiever engine. This is a user who decides when to start the new IOC. We want to avoid running the same IOC in two various versions and users receiving alarms from the killed, old IOC version during deployment.

MONITORING

System and application monitoring is a vital IT function that has a wide range of benefits for various businesses and facilities. It can save money on network performance, em-

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

ployee productivity, and infrastructure costs, and it is far more strategic than its name implies [8].

The monitoring can be divided into basic types depending on their purpose:

1. Availability monitoring, which concerns server management and services status;
2. Web and network performance monitoring, which are tools that can detect excessive network load, a high number of requests to a single service, or any network bottleneck; and
3. Application monitoring, which tracks the performance of an application and spots any issues before they cause service failure [9].

An effective monitoring system has many features from which developers and infrastructure users can profit. Monitoring services can not only prevent incidents but also allow you to detect them faster when they do happen. Fast incident detection results in time and money savings. Monitoring servers and systems improve the use of the hardware [8]. We can take advantage of all hardware resources if we know that they are in the operational state. We can predict network traffic or excessive CPU usage by some of the applications or services and try to optimize hardware architecture. In terms of early issue detection, it is important to provide a real-time notification [8]. They will not only alert you about performance issues but also make it easy for you to resolve those issues. It can be done via emails, text messages, or any other integration with communication applications (e.g. Slack). Finally, effective monitoring enhances the end-user experience. If infrastructure is slow, it will result in many support calls from end users. Delivering top-of-the-line IT-enabled services can improve the productivity of the systems and increase the number of satisfied end users [8, 10].

Laser Monitoring System

The designed monitoring system is composed of many elements for both servers and IOC monitoring. We have created the custom Go client that uses CA protocol (using the specially designed Go implementation of EPICS CA library) that monitors PVs from the IOC. We focused on PVs from the IocStats module like IOC CPU, RAM usage and Heartbeat, which informs about IOC state, but also PVs that inform about the connection state between IOC and the device (asynRecord) and some IOC-specific PVs like the number of acquired frames from camera IOC. The Go client exposes metrics about the number of monitoring PVs and their state (connected/disconnected).

The client sends data to VictoriaMetrics. It is an open-source data series database typically used for processing high volumes of data and for long-term data storage [11]. It uses 10x less RAM than InfluxDB and up to 7x less RAM than Prometheus, Thanos, or Cortex when dealing with millions of unique time series. It can be used as a drop-in replacement for Prometheus because it supports Prometheus querying API [11].

For visualisation, we chose yet another open-source tool - Grafana. It helps to create queries and visualize application performance metrics with customized data. The tool lets one create monitoring dashboards for metrics over a specific period, so it can easily be adapted for a specific project [12].

To monitor the hardware, we decided to use the Prometheus node exporter, which exposes metrics with all essential server parameters. Those are scraped directly to VictoriaMetrics and visualized on the available dashboard templates prepared by the community.

The alerting system is based on the Prometheus Alertmanager and vmalert. We prepared a set of alert rules in PromQl about hardware metrics and IOC metrics - for example, an alert should be raised when the IOC heartbeat stops growing or when the PV with the device connection state changes its value. To provide real-time messages we redirected alarms to one of the Slack channels. In the error message, one can see the level of the alert (error, warning, resolved), the metric name and its value, the host origin, and a short metric description.

DOCUMENTATION

Documentation is an essential part of building control systems, regardless of whether the build is based on remote working or a traditional build. In designing the documentation, we have tried to follow current trends to ensure is easily accessible, is up-to-date, covers as much context as possible, and is easily maintainable and modifiable in the future.

Key elements of the system have been recorded in the form of Architecture Decision Record (ADR) documents, which contain important architectural decisions together with context and consequences. Each document has a number and relates to the one problem it describes. Examples of such documents include those describing the network in the system or the EPICS PV naming scheme.

Another important documentation element is a network map with addresses and information on the location of individual servers and computers used to operate the system. All logins and passwords are stored in a password manager.

A document containing information on each device used in the system is also essential. This includes information on the device, the manual, and contact with the people responsible for the software on the supplier end.

Information about what software is running on which server is contained in the deployment configuration files, which are an integral part of the main repository with the system code. Hence, additional documentation of these is not necessary.

CONCLUSIONS

We have successfully delivered the fully operational infrastructure for the 100 Hz laser control system. It follows modern management practices (CI/CD, IaC, multiple environments) and limits human intervention to a minimum by general automation of the processes. Automated IOC testing, together with unmanned deployment to the staging environment, makes the control system less prone to errors and allows for fast integration with existing applications.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH011

Thanks to the monitoring that covers not only server parameters but also IOC statistics, we can effectively react to failures and try to prevent them. We have proved that the research infrastructure can be similar to state-of-the-art bank applications or web applications in terms of testing, deployment, and monitoring.

There are still areas that we are working on and trying to improve. We are currently developing our implementation of the device simulators. Since the internal state of the device is not a crucial part of the simulators from a testing point of view, we decided to limit that aspect and focus on easier protocol implementation that would allow a user to describe any non-standard communication type. This will make IOC testing less time-consuming and more effective. What is more, together with laser scientists, we are developing more user-friendly monitoring dashboards, benefiting from their knowledge and experience.

ACKNOWLEDGEMENTS

The 100 Hz laser system was funded by European Regional Development Fund and the state budget of the Czech Republic project HiLASE CoE (CZ.02.1.01/0.0/0.0/15-006/0000674); Horizon 2020 Framework Programme (H2020) (739573).

REFERENCES

[1] *Why should you use Staging Environment?*, https://apiumacademy.com/blog/why-should-you-use-staging-environment/ (accessed on 2022-09-20)

[2] M. Kraimer, J. Anderson, A. Johnson, W. Norum, J. Hill, R. Lange, B. Franksen, P. Denison, and M. Davidsaver, "EPICS Application Developer's Guide", EPICS Base Release 3.15.5., https://epics.anl.gov/base/R3-15/5-docs/AppDevGuide.pdf (accessed on 2022-09-20)

[3] EPICS about, https://epics-controls.org/about-epics/ (accessed on 2022-09-20)

[4] Lewis documentation, https://lewis.readthedocs.io/en/latest/index.html (accessed on 2022-09-20)

[5] A. Axelrod, "Complete Guide to Test Automation", 1st ed. edition, Apress, 2018, pp. 3-4.

[6] M. Labourady, "Pipeline as Code: Continuous Delivery with Jenkins, Kubernetes, and Terrafom", Manning, 2021, pp. 3-5.

[7] M. R. Pratama and D. Sulistiyo Kusumo, "Implementation of Continuous Integration and Continuous Delivery (CI/CD) on Automatic Performance Testing", 2021 9th International Conference on Information and Communication Technology (ICoICT), 2021, pp. 230-235.

[8] S. Lagus, "Effective Monitoring and Alerting: For Web Operation", 1st edition, O'Reilly Media, 2012, pp. 1–5.

[9] *What Is IT Monitoring?*, https://www.splunk.com/en_us/data-insider/what-is-it-monitoring.html (accessed on 2022-09-20)

[10] J. Hernantes, G. Gallardo, and N. Serrano, "IT Infrastructure-Monitoring Tools", *IEEE Software*, vol. 32, no. 4, pp. 88-93, July-Aug. 2015. doi:10.1109/MS.2015.96

[11] VictoriaMetrics documentation, https://docs.victoriametrics.com/ (accessed on 2022-09-20)

[12] Grafana documentation, https://grafana.com/docs/ (accessed on 2022-09-20)

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH013

UPGRADING THE IRRAD CONTROL SYSTEM GUIs USING OPEN-LICENSE AND CROSS-PLATFORM TECHNOLOGIES*

B. Gkotse†[1], A. Abdulhalim, A. S. Mølholm, F. Ravotti
Experimental Physics Department, CERN, Geneva, Switzerland
P. Jouvelot, Mines Paris, PSL University, Paris, France
[1]also at Mines Paris, PSL University, Paris, France

Abstract

The CERN Proton Irradiation Facility (IRRAD) is a reference facility in high-energy physics for the qualification of detectors, materials, and electronic components against radiation. A proton beam with a momentum of 24 GeV/c is delivered from the CERN PS accelerator to IRRAD and impinges on the components being tested, placed on remotely controlled movable stages. This equipment, operated by dedicated control systems, allows for the precise positioning of components in or out of the beam and facilitates the handling of irradiated components, while minimising the radiation received by the IRRAD operators.

Originally, the implementation of the Graphical User Interfaces (GUIs) of the IRRAD control system was based on proprietary software, thus limiting it to specific operating system. To address the issues linked to such dependencies in terms of openness, ease of development and access to state-of-the-art technologies, new GUIs have been designed and developed with open-license cross-platform software. In this paper, the IRRAD control system software architecture is detailed, and the lessons learned while implementing these new feature-rich GUIs are presented.

INTRODUCTION

The Proton Irradiation Facility at CERN (IRRAD) is an infrastructure dedicated to performing radiation hardness testing (irradiation experiments) on detectors, electronics, and materials. The IRRAD facility, located in the East Area of the CERN accelerator complex, receives a proton beam from the Proton Synchrotron (PS) accelerator with a momentum of 24 GeV/c in spills of 400 ms and a Gaussian shape typically 12×12 mm^2 FWHM wide [1]. The components that need to be tested (Devices Under Test, or samples) can be placed along the beam trajectory on the top of nine remotely movable stages, called IRRAD tables. The IRRAD tables have three degrees of freedom and can be used to move the samples horizontally (x-axis), vertically (y-axis) or rotate them with an angle (θ) with respect to the beam axis. These tables allow for positioning samples in a volume of up to $20 \times 20 \times 50$ cm^3 in and out of beam, or performing a scan (e.g. asynchronously move the samples across the beam direction in order to extend the irradiated portion of the samples). Placing or removing samples on the IRRAD tables requires accessing the irradiation area, which can be done only once per week, when the beam is stopped. However, for smaller samples with dimensions up to $5 \times 5 \times 20$ cm^3, a conveyor system (shuttle) can be used; this can move samples from the outside to the irradiation zone, following a 9m-path without the need of stopping the beam.

During CERN Long Shutdown 1 (2012-2014), a new hardware infrastructure had been put in place at IRRAD and software Graphical User Interfaces (GUIs) had been built, based on CERN-supported proprietary software, for the control of its tables and shuttle. These GUIs were sufficient for operating during CERN Run 2 (2014-2018) [2]. Nevertheless, several restrictions on portability, dependencies on specific platforms and requests for additional functionalities have since deemed necessary the upgrade of the control system GUIs using current open-license and cross-platform technologies. Presenting and discussing this important transition is the subject of this paper.

This article first provides an overview of the hardware components and infrastructure used in the IRRAD control systems. Then, we describe the open-licence and cross-platform technologies used for the development of the new Graphical User Interfaces (GUIs), explaining our software choices and architecture. Details of the new functionalities, software architecture and database schemes are provided for both IRRAD tables and shuttle control systems. Finally, we discuss the lessons learned during this software transition, before concluding and introducing future work.

IRRAD CONTROL SYSTEMS HARDWARE

Since the operation of the IRRAD equipment happens in a radiation environment, the hardware chosen for the tables, shuttle and associated control systems is custom-made, had to ensure radiation tolerance and be easily customizable depending on the experimental user requests. For both systems, some common components have been used but still certain differences remain. Details about the hardware infrastructure are provided in the following paragraphs.

IRRAD Tables

Each IRRAD table uses two stepper motors, one for horizontal movement and one for the rotating axis, while an AC motor is used for vertical movements. Figure 1 shows 3 of the IRRAD tables of the first zone of irradiation in the IRRAD facility. The three motors are controlled using an M300 microprocessor. The communication with the microprocessor is performed through the RS232 serial protocol. Since there are nine of these tables that require this type

* This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement no. 654168.
† Blerina.Gkotse@cern.ch

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH013

of communication, Ethernet-to-Serial devices have been installed in the facility, providing multiple RS232 serial ports. This allows computers running the control system software to communicate through multiple virtual COM ports. One control box per IRRAD table containing the microprocessor and control buttons is installed in the IRRAD control room for manually controlling the tables. However, this particular setting requires the presence of operators in the control room and induces some limitations on the type of actions that can be performed.

Figure 1: Irradiation tables (front) and shuttle system (back) in the IRRAD irradiation zone.

Shuttle System

The shuttle system is controlled through two separate motors. The first one is used for the movement of the shuttle over the 9m-long path in or out of the irradiation zone, referred to be the y-axis. The second motor is used for moving the shuttle in or out of the beam trajectory, named x-axis. Figure 2 shows the shuttle outside of the irradiation area and illustrates the two axis it can be moved along. The y-axis uses an AKD Kollmorgen driver [3], and the communication is performed through the telnet protocol. The x-axis has a stepper motor controlled by a M300 microprocessor, and the RS232 serial communication protocol is used, as for the control of IRRAD tables.

SOFTWARE TECHNOLOGY AND ARCHITECTURE

The software technologies used in IRRAD have to be compatible with the hardware located in the facility. Moreover, since IRRAD is a small-scale infrastructure with limited manpower and software expertise, some lightweight and easily maintainable control-software solutions are required. In the first IRRAD run, the control system GUIs that were used were developed using Windows Forms [4] and coded in C++ and C#. Even though the interfaces were considered operational and user-friendly, limitations and dependencies on specific operating system and proprietary software presented some drawbacks (see Section Discussion), which led us to consider upgrading the IRRAD control software.

Figure 2: Shuttle system, part outside of the irradiation area, where the samples are loaded.

Given the fact that the CERN community widely uses LabVIEW [5], SCADA [6] and WinCC-AO [7] for control systems, they were first considered as candidate solutions for our work. However, dependencies on industrial software, not easily maintained by a small team, were a challenge not possible to overcome. Other free and open-source software tools were also discussed such as EPICS [8] and Tango [9]. However, their communities are rather limited, since they are used mainly in high-energy physics, making their learning curve, in the presence of scarce support, too steep. A more lightweight solution was thus considered.

In the following paragraphs, we describe the software technology chosen to upgrade the IRRAD control system architecture and the motivations behind those choices.

PyQt

The PyQt library [10] includes a set of python bindings for the QT application development platform [11]. More specifically, for the IRRAD GUIs use case, PyQt5, released under GNU General Public License (GPL) v3 license, was used. This choice was not only made because PyQt is a free software, but also because it combines the flexibility of Qt for developing fast and interactive user interfaces and the coding simplicity and effectiveness of python.

pySerial

As mentioned in the previous section, the communication with the M300 microprocessor has to be performed serially using the RS232 protocol. For this purpose, pySe-

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH013

rial, which is a python module for serial communication management [12], was used.

MySQL

A back-end database was needed for storing several configuration parameters and position history for the GUIs. The well-known open-source MySQL framework was deemed the relational database of choice to host these data [13]. The IRRAD-dedicated database instance is hosted on the Database-on-Demand (DBOD) infrastructure provided by CERN, which allows for an easy back-up and maintenance plan, including regular upgrades [14]. However, the GUIs can be easily configured with a different database, if need be.

Software Architecture

A Model-View-Controller architectural software approach has been adopted for the development of the GUIs, while the application communicates with the SQL database in the back end.

IRRAD TABLE CONTROL SYSTEM GUI

The main requirements for the IRRAD Table GUIs were to allow users to move the samples easily and safely in the required positions. Other important requirements are that users should be able to monitor visually their actions, aware of the position limits and able to intervene any moment by stopping their actions. Moreover, a history of the performed movements and positions selected needed to be kept for logging purposes.

Functionalities

The above mentioned requirements were then translated into the functionalities described in the following items:

- Setting hardware parameters about each motor (e.g., resolution, screw dimensions, offset, etc.);
- Setting communication parameters (e.g., baud rate, parity, COM port, etc.);
- Configuring the sample position, in mm, in the microprocessor memory (since the microprocessor memory is limited, only five specific positions can be configured: park, center, left, right and one that is used for custom positions);
- Moving the IRRAD tables in the configured positions by sending the corresponding commands to the microprocessor;
- Visualising the position;
- Calibrating the motors (since stepper and AC motors are used for the IRRAD tables, a dedicated process for finding the proper correspondence between step numbers and distance in mm, which depends on hardware characteristics such as screw dimensions, is used. For stepper motors, this consists in finding and setting equal to zero the position of a reference switch placed at 0 mm distance and then also placing the motor to the maximum position in mm and with the maximum number of steps).

The user can select to have the full views of the three axes displayed, as shown in Fig. 3, or only one at a time.

Database

Dedicated database tables have been designed and used to save data deemed key for the operation of the GUI and log positions and movements. More specifically, the database tables contain information about the following elements.

Stepper Motor These data contain the stepper motor's hardware settings such as calibration information, resolution, screw dimensions, maximum position, etc.

AC Motor Since AC motors have different characteristics than the stepper motors, a dedicated database table is used for them.

Motor This table contains the names and the motor numbers and is linked to the two previous tables through foreign keys.

Custom Position Custom positions are also stored in the database for each table. In this way, a larger set of custom positions can be saved and used, overcoming the microprocessor memory limitations, by creating visually more buttons with different positions. Nevertheless, in the background, the memory used for storing the custom position is overwritten each time.

Movement The movements are logged in the system for tracing performed actions.

Full documentation and manual for the system can be also found online [15, 16].

IRRAD SHUTTLE CONTROL SYSTEM GUI

Operational safety, precision and usability were the most important requirements for the development of the shuttle GUI, given its operation in a radiation environment.

Functionalities

The main functionalities for the shuttle system are:

- Moving the shuttle in the defined position (Reference, Loading samples, Park and Beam positions);
- Constraining the users on performing certain movements that could affect the precision of the movement (for example, the user should not be allowed to perform certain actions such as going to the Beam position once the shuttle has moved backwards; in that case, the shuttle should move to the Reference position before moving to Beam);
- Monitoring the activity, using two AD6 monitors placed in the Load and Park position in order to comply with safety-related procedures (the data of these monitors are logged and should be controlled before moving the shuttle in the Load position);

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH013

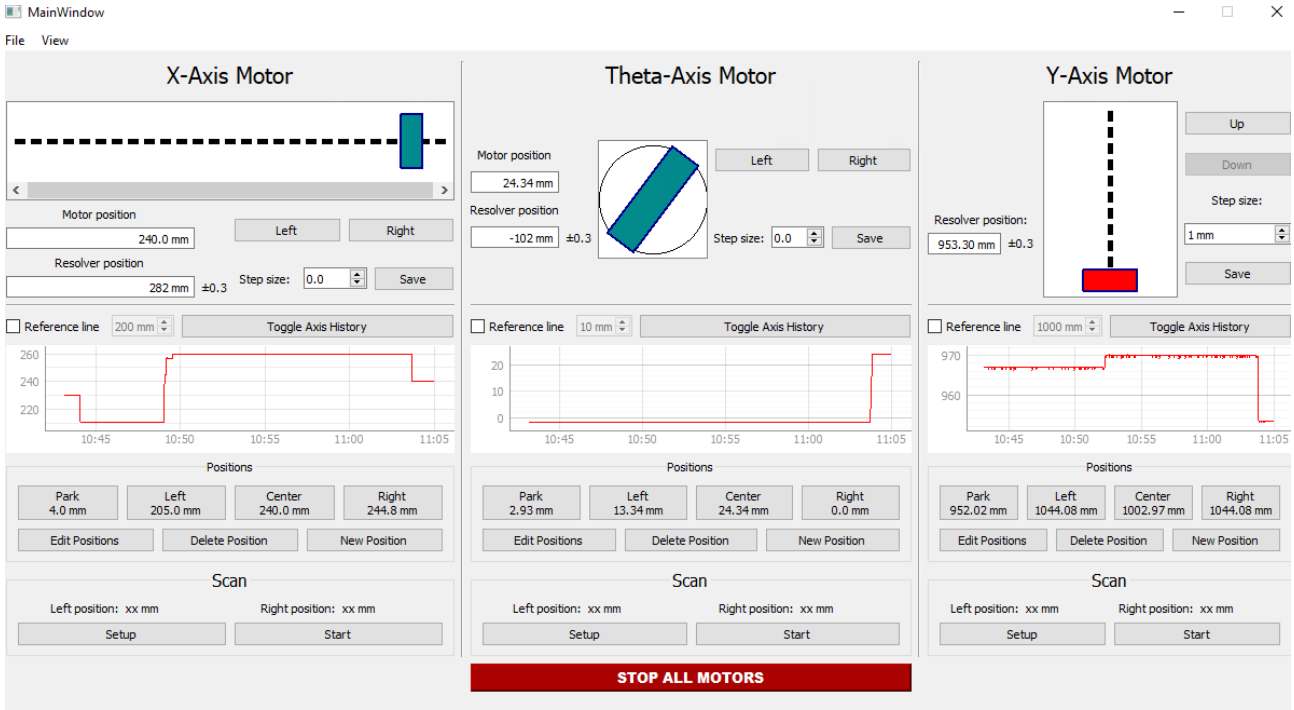


Figure 3: IRRAD Tables Control System GUI.

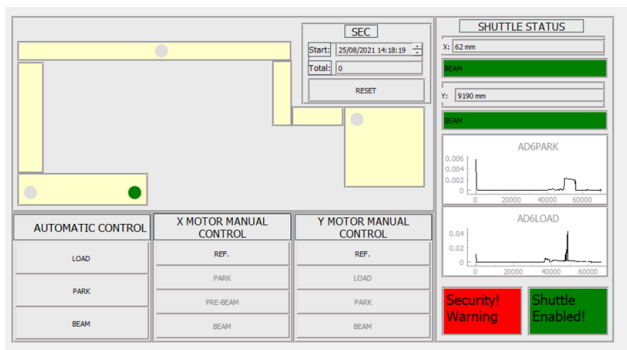


Figure 4: Shuttle GUI.

- Implementing software interlocks, e.g., for not allowing the user to move the table in the Load position if the activity monitored by the AD6 is not lower than a specific limit set by the Radiation Protection authority.
- Visualising on a display the shuttle path (moreover, activity diagrams have been integrated in the interface);
- Monitoring the cumulated proton intensity when the shuttle is in beam position;

A screenshot of the shuttle interface is shown in Fig. 4.

Database

Currently, the data stored in the database for the shuttle system contains the movement history and the cumulated proton intensity for each irradiation run.

Further details and documentation can be also found online [17].

DISCUSSION

Even though IRRAD is a small-scale facility and is thus supported by few software developers, moving its experiment-management software from proprietary to more lightweight, free, open-source and cross-platform technologies turned out to not be very complicated or time consuming. It took a team of two short-term interns under the supervision of one PhD candidate and eight months to come to some workable solutions. The major challenges consisted of designing and implementing advanced functionalities for a robust, safe and user-friendly system that could be easily customisable also by non software-experts. Based on a broad and open-source community and proper documentation [10], finding solutions when problems arose was relatively easy and fast.

Comparing the old C++/C# based and new python-based environments in terms of size, the present choice led to a significant decrease in the number of lines of code. For instance, for the IRRAD Table GUI, we went from about 12,300 lines of code to only 5,500 lines; regarding the Shuttle GUI, the respective numbers are 2,180 and 1,730. This allows for better readability and faster maintenance in case of need. Also, having based the new development on the python language provides the opportunity to extend more easily the code by importing advanced modules and libraries developed by the python community, which is moreover growing larger and larger. Finally, using more recent technologies such as python helps also to find personnel who have the requested software skills and can be trained to work on the project more easily and faster.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH013

In addition, the technology upgrade performed by the IRRAD team is in sync with the CERN IT policy that recommends moving towards open-license software as part of the the license-management MALT project [18], which encouraged the search for alternative free and open-source technology solutions to proprietary ones in CERN projects.

Globally, these technologies have increased the portability and the ease of installation of the GUIs on different operating systems. In comparison to more advanced and industrial control system GUI software frameworks such as WinCC-AO [7] , these interfaces are more lightweight. They seem more suitable for small-scale experiments and infrastructures that can be easily used and maintained by non-IT experts.

Finally, these upgrades were also in line with the needs of the IRRAD facility to cope with a larger demand of user experiments. Traceability and precision were enhanced by the newly introduced logging functionalities such as storing the table movements in a database.

CONCLUSION AND FUTURE WORK

This paper describes the new GUIs developed for the control systems of the IRRAD proton irradiation facility. The hardware components and their uses have been described. For the software development, free, open-source and cross-platform software technologies such as PyQt, pySerial and MySQL have been used and the advantages of this move from proprietary software discussed.

In parallel to these developments, the IRRAD team is working on new beam-monitor detectors and data acquisition systems. These detectors will be installed on the IRRAD tables and used to detect when the tables are exposed to the beam. Therefore, as future work, we envision that the data acquired from these systems could be used to operate a system that would automatically move the tables depending on the beam conditions.

Another future work could include designing and implementing some web interfaces. Since the back end is already developed using python, a python web framework such as Django [19] could be used for this kind of development. In that case, IRRAD system access could be enabled outside of the CERN facility. Therefore, ensuring IT security would be a crucial factor.

REFERENCES

[1] F. Ravotti, B. Gkotse, M. Moll, and M. Glaser, "IRRAD: The New 24 GeV/c Proton Irradiation Facility at CERN", in *Proc. AccApp'15*, Washington, DC, USA, Nov. 2015, pp. 182-187. http://accapp15.org/wp-content/data/index.html

[2] B. Gkotse, M. Glaser, P. Jouvelot, E. Matli, G. Pezzullo, and F. Ravotti, "Towards a Unified Environmental Monitoring, Control and Data Management System for Irradiation Facilities: the CERN IRRAD Use Case", in *Proc. RADECS 2017*, Geneva, Switzerland, Oct. 2017, pp. 1-8. doi:10.1109/RADECS.2017.8696209

[3] AKD Kollmorgen website, https://www.kollmorgen.com/en-us/developer-network/akd-drive/

[4] Windows Form documentation, https://docs.microsoft.com/en-us/dotnet/desktop/winforms/

[5] LabVIEW website, https://www.ni.com/it-it/shop/labview.html

[6] A. Daneels and W. Salter, "What Is SCADA?", in *Proc. International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'99)*, Trieste, Italy, 1999, pp. 339-343. https://jacow.org/ica99/papers/mc1i01.pdf

[7] WinCC-OA Service website, https://readthedocs.web.cern.ch/display/ICKB/WinCC-OA+Service/

[8] EPICS website, https://epics-controls.org/

[9] TANGO website, https://www.tango-controls.org/

[10] PyQt website, https://pypi.org/project/PyQt5/

[11] Qt website, https://www.qt.io/

[12] PySerial documentation website, https://pyserial.readthedocs.io/

[13] MySQL website, https://www.mysql.com

[14] Database-on-Demand website, https://dbod-user-guide.web.cern.ch/

[15] A. S. Mølholm, B. Gkotse, and F. Ravotti, "Python IRRAD Motor Control Application (PIMCA):How it works", CERN, Geneva, AIDA-2020-NOTE-2021-003, 2021, https://cds.cern.ch/record/2750195

[16] A. S. Mølholm, B. Gkotse, and F. Ravotti, "Python IRRAD Motor Control Application (PIMCA):How to use", CERN, Geneva, AIDA-2020-NOTE-2021-002, 2021, https://cds.cern.ch/record/2749936

[17] A. Abdulhalim, "GUI implementation for Controlling and Monitoring of the IRRAD Shuttle System", CERN, Geneva, 2021, https://cds.cern.ch/record/2779934

[18] MALT project website, https://malt.web.cern.ch/

[19] Django website, https://www.djangoproject.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH014

HOW GANIL PLAN TO USE WEB TECHNOLOGIES TO UPDATE THE CONTROL SYSTEM USER INTERFACE

C.H. Haquin†, O. Delahaye, C.H. Patard, F. Pillon, J. Pivard, G. Senecal, GANIL, Caen, France

Abstract

The Grand Accelerateur National d'Ions Lourds (GANIL) Control System was developed in the first half of the nineties with ADA language. The user interfaces use MOTIF and XRT widgets. User interfaces have become more and more difficult to modify and there is a risk of obsolescence. GANIL plan to replace these old technologies and web technologies are anticipated. This paper will present the strategy defined to make the switch.

CURRENT SYSTEM

The control system of the original GANIL accelerator (2 ions sources, 5 cyclotrons), called GANICIEL, relies on several software (about 65), entirely developed in ADA language specifically for this machine thirty years ago, by the developers who were present at that time.

There are two main software layers (see Fig. 1), the first one, the human machine interfaces layer (about 45 software), corresponds to all the software required to tune the accelerator by operators from the control room. The other, the equipment control layer, comprehends all the real-time servers (about 10 software multi instantiated on 31 VME) interacting with equipment to handle Human Machine Interface's (HMI) requests and ensure parameters setting and data monitoring.

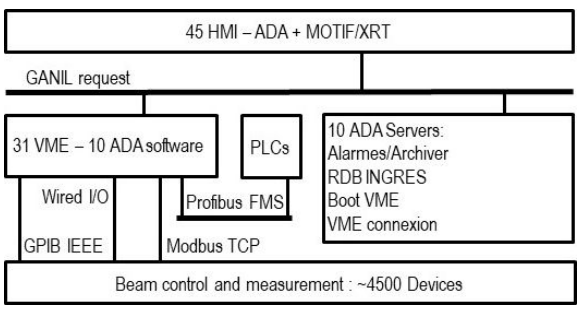


Figure 1: Current GANIL control system architecture.

To operate, both layers rely on relational databases (INGRES technology), at HMI layer level, widget are powered by MOTIF/XRT technology, at the equipment control layer level, VxWorks 6.9 operating system is required to run VME software.

In between the two layers, there is a batch (about 10) of software servers providing services for data bases access, alarm monitoring, data archiving and minimization of the Transmission Control Protocol/Internet Protocol (TCP/IP) connection between HMI and Versa Module Europa (VME).

GANICIEL Request Concept

The communication between the HMI layer and the equipment control layer is achieved over TCP/IP with a homemade request system providing simple and complex order services. Simple order offer the possibility to interact quickly with an equipment by sending unitary order with no reply, complex order gives the possibility to execute a sequence of order on an equipment and receive a report with only one transaction between a HMI and a VME.

Technical Threats and Opportunities

ADA Language The ADA language appeared in the early eighteen with ADA83 and was enhanced with revision ADA95, ADA2005 and ADA2012. Software developed in ADA are renowned to be robust, its attractiveness increased with the needs for security for payment systems, ADA is a technology we can rely on for the future. However, it is not widespread, the lack of skilled people and the weak attraction of the young developers for it are the main drawback that pushes us to reduce its use. Currently there are almost 3 MLoC in ADA for the whole control system (see Table 1).

Table 1: ADA Source Code Distribution

	Million Lines of Code
HMI, graphical rendering	1,0
HMI, command/algorithm	0,5
Total for HMI layer	1,5
VME ADA software	0,3
ADA servers	0,4
Total	2,7

HMI Layer There are two kind of HMI, one dedicated to equipment command and one to the so-called algorithmic command, equipment commands correspond the simplest HMI dedicated to a single equipment like basic Input/Output (I/O), driving a power supply, insertions. Algorithmic command HMI implement complex sequences like cycling a cyclotron or optimizing the beam transmission, they can involve several equipment potentially spread-out on several crates.

There are 1.5 Million Lines of Code (MLoC) for the HMI layer, 1 MLoC for the graphical rendering, the remaining half MLoC for the commands and algorithm. The widgets used for the graphical rendering are issued from the MOTIF/XRT libraries which are not commercialised and maintained since 2016, workmate that are now retired managed to keep these libraries up and running at each hosts renewal (every five years). We do not have the MOTIF/XRT skills anymore, there is a risk that we don't manage to repeat the performance for the next host renewal (2026) and we may face a blocking software or hardware incompatibility.

Equipment Control Layer The equipment control layer has a typical early nineteen architecture based on 31 VME crates allowing the control of the 4500 devices of

† christophe.haquin@ganil.fr

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH014

the accelerator. Each crate consists of a Central Processing Unit (CPU) card plus a set of I/O cards, there are 230 digital I/O, analog I/O and protocol card, some of the I/O cards were homemade, some are off the shelf products, protocol cards are off the shelf but execute a closed source software. Each crates executes one instance of the 10 VME software application, at boot time, each instance retrieves from a data base sever its configuration describing the set of I/O cards installed in the crate, the I/O mapping of each card and the names of the equipment to be controlled by that crate.

The equipment control layer works fine, unlike the HMI layer, there is no urgent risk related to obsolescence. However, according to the number of devices that are now accessible over TCP/IP network and the performances of the hypervisors, the CPU/protocol cards association could be replaced by a virtual server, a VME crate with its CPU card and batch of I/O cards could, in many cases, advantageously be replaced by connected I/O systems. In other words, the innovations of the last decades give us the opportunity to reduce the cost related to spare and repair, ease interventions and maintenance with remote access and improve service monitoring.

THE RENOVATION PROJECT

In 2023 the GANIL will celebrate the 40th anniversary of the first beam, and the demand from the physicist remain high, the GANIL must consider its exploitation for at least the next two decades. Consequently, an ambitious renovation project is currently being defined in which we plan to solve the obsolescence risk and modernize the architecture of the control system with latest technologies.

Staff and Skills Considerations

The main objective for the control system renovation is to eliminate the obsolescence risk, but, skills and human resources availability have also been taken into account to design the renovation plan.

The current system is the second-generation control system, it was designed and developed by eight people from GANIL between 1990 and 1992, the commissioning occurred in 1993, it has been enhanced during the next two decades by its creators, there were up to 10 people working on it. Things started to become more complicated in the last decade when people retired and not all of them were replaced, currently only 4 young permanent people helped by 3 non-permanent people and they have to work not only on the legacy machine but also on the Experimental Physics and Industrial Control System (EPICS) control system for the Systeme de Production d'Ions Radioactifs Accelere en Ligne2 (SPIRAL2) Linear Accelerator (LINAC), Super Seperateur Spectrometre (S3), Désintégration, Excitation et Stockage d'Ions Radioactifs (DESIR) and New GANIL Injector (NewGain). So, the challenge we are facing is how can we do more with less people?

Though the current control system works fine it still need to evolve constantly, modifications take too much time because the tools are basic, it is difficult to find ADA developers (student or a temporary contract or a software service company) or to motivate young developers with this language.

Consequently, the renovation plan aims at solving this obsolescence risk with XRT/MOTIF technology but also to reduce the human resources needs for maintenance and open wide outsourcing alternative.

HMI Layer Modification

The HMI layer was developed at a time where software quality consideration like specification, maintainability, scalability, testability were not as obvious and rooted as today, the architecture of each HMI is not really layered, the view and the control are completely interleaved.

We thought about the possibility to redevelop all the HMI but we came to the conclusion that though the actual code is far from being perfect, the code in which there are algorithms and intelligent control, state machines or sequences still has a lot of value because of its 30 years of maturity. We defined a two steps strategy aiming at preserving and building on this value.

The first step: the refactoring of the code prior removing the XRT/MOTIF stack, is an unavoidable prerequisite in order to start from a sound basis with two well-defined view/control layers interacting with a well-defined interface. This step gives us the possibility to validate that the refactored HMI work like the non-refactored ones and if not, the guarantee that the root cause is in the refactored code and not related to the introduction of the new technology.

The second step aims at minimising the impact on the valuable code by replacing only the view layer with theoretically no impact on the control layer or at least minimize the modification and avoid that each HMI has its own set of specificity and workaround.

The Web Technology Introduction The most straight forward strategy would have been to replace XRT/MOTIF by gtkADA but, according to the staff & skills consideration, we decided that we should choose a more widespread technology, that's how we started to think about using web technology.

Though no framework is chosen yet, we redefined the control system architecture by introducing the web concepts (see Fig. 2).

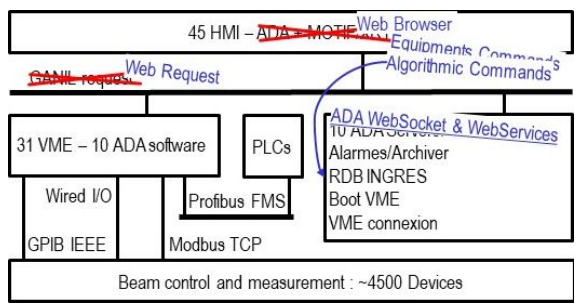


Figure 2: HMI Modification.

The strictly speaking user interfaces will be accessible with a web browser, the GANIL request system will be replaced by WebSocket [1] for commands that require full-duplex communication and Web-Services for the other.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-TH014

Those web communications will provide access to the View/Control interface defined during the refactoring in order to make available for the user interface the ADA most valuable services. Finally, the ADA code corresponding to the control layer and all the other server will migrate to web server.

Equipment Control Layer Modification

As mentioned before, to ease the maintenance, the number of VME crates could be dramatically reduced, we estimate that 12 crates can be directly virtualized since they are dedicated to beam profiler, power supply and PLCs accessible with Modbus/TCP protocol, 10 more crates with off the shelf I/O cards could be virtualized by introducing connected I/O devices, this requires to develop new drivers. Finally, 9 crates with homemade I/O cards will remain unchanged (Fig. 3). Virtualizing those crates would impact heavily the controlled devices and imply import hardware and software development to finally obtain the same functionalities. This is considered not worth the effort.

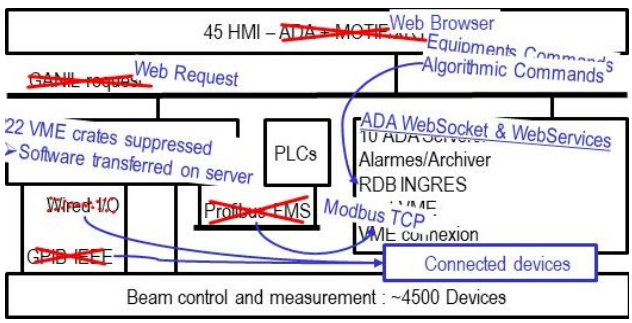


Figure 3: Equipment control layer modification.

As for the HMI control layer ADA code, the code transferred from the VME will integrate the batch of web services and WebSocket services will have to be activated in the 9 remaining VME (Fig. 4).

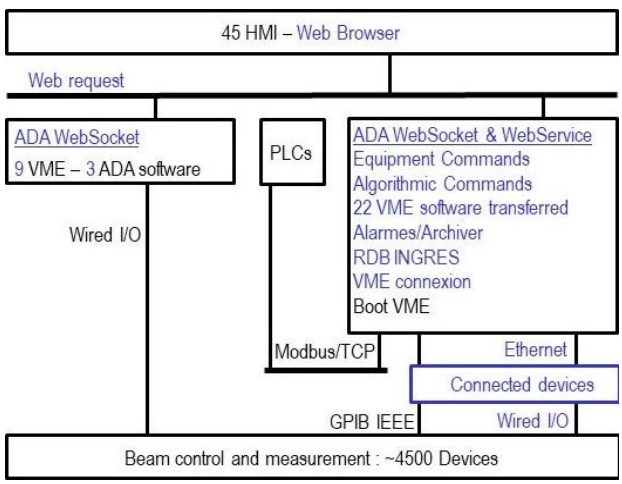


Figure 4: Final GANIL control system architecture.

How PLC Technology Could Also Be Involved

We've also taken into account the technical improvement that are coming from the PLCs. With the current architecture, 3 VME and 10 HMI have no real added value they are just gateway to PLCs, this is legacy from the time where supervision was available only on windows platform and not on Linux platform. Since there is also a renovation plan for PLCs, old Siemens series 5 will be replaced by S7 series, we are also looking closely to WinCC Unified, the new Siemens HMI technology, to replace the 10 ADA HMI and remove the 3 VME crates or their virtualized version.

Human Resources

The effort was first estimated for a well-known and reasonably complex HMI, then we applied the resulting person-years/MLoC ratio to the other HMI, we finally obtained a total of seven person-years for the whole HMI layer modification.

The corresponding method was applied to a VME taking into account the integration of the connected I/O devices, we obtained a total of five person-years.

The GANIL control system renovation is therefore estimated at twelve person-years and we plan to outsource seven person-years.

Planning

We expect the project kick-off at beginning of 2023, the first task will be to establish the outsourcing call for tender and we plan the technical tasks to be executed between 2024 and 2026.

CONCLUSION

How can we replace the old XRT/MOTIF stack has been a discussion topic in the software team since at least fifteen years! Discussions often ended up with the conclusion that it was not possible without rewriting everything and we didn't had time and resources.

The rise of an ambitious renovation project at GANIL scale gives us the opportunity to obtain the resources required by such a complex task and make the change come true.

Moreover, with the outsourcing approach and the will of the young developers to work with state of the art technologies and engineering process, it open the perspective to introduce better test & validation practices and use of continuous integration/deployment (CI/CD) tools.

Finally, lessons need to be learnt from the commissioning of the SPIRAL2 accelerator in order to tackle HMI development not only as a software development task but as a holistic product [2] requiring the engagement of the users. From tuning capabilities point of view, current GANIL HMI are fully satisfying, this will be the functional baseline to be integrated in a user centric process based on regular delivery and demo, taking into account design and user interface/experience (UI/UX) considerations from the start.

REFERENCES

[1] A. Uchiyama *et al.,* "EPICS Channel Access Using WebSocket", in *Proc. PAaPAC'12*, Kolkata, India, 2012, paper WECC02, pp. 7-9.

[2] M. Cagan, *Inspired: How to Create Tech Products Customers Love*, Silicon Valley Product Group, USA: Wiley, 2017.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP1

INAU: A CUSTOM BUILD-AND-DEPLOY TOOL BASED ON Git

L. Pivetta*, A.I. Bogani†, Elettra Sincrotrone Trieste, Basovizza (TS), Italy

Abstract

Elettra Sincrotrone Trieste is currently operating two light sources, Elettra, a third generation synchrotron, and FERMI, a free electron laser. Control systems are based on a number of diverse systems, such as VME-based front-end computers, small embedded systems, high performance rack-mount servers and control room workstations. Custom device drivers and hard real-time applications have been developed during the years, exploiting the technologies adopted such as RTAI and Adeos/Xenomai, which make a massive update demanding. Modern CI/CD tools are then not available for legacy platforms, and a custom tool, integrating git and a database back-end to build and deploy software components based on release tags has been developed.

INTRODUCTION

At Elettra and FERMI some complex subsystems, such as the global orbit feedback or the hard real-time interfacement of sensors and actuators, have been implemented exploiting the capabilities of GNU/Linux together with hard real-time extensions. Moreover, most of the interfacement of legacy equipment is made by VME based digital and analog I/O boards running on PowerPC single board computers. A number of device drivers and hard real-time applications have been developed in the years, making a complete system upgrade unfeasible. This turns into an effective limitation against adopting and using modern Ci/CD tools and procedures, which are not compatible with legacy systems.

LEGACY SYSTEMS, AND NOT

Different platforms are in use at Elettra and FERMI, depending on generation and the specific integration requirement. Leaving out some oldest systems running Microware OS-9, which are no more developed, and thus, not supported by INAU, several generations of GNU/Linux based single board computers and rack-mount servers are in use in front-end systems:

- Emerson MVME5100 and MVME6100, running Debian 3 (Linux kernel 2.4)
- Emerson MVME7100, running Ubuntu 7.10, (Linux kernel 2.6)
- Supermicro Intel-based rack-mount servers, running Ubuntu 10.04 (Linux kernel 2.6)
- Supermicro Intel-based rack-mount servers, running Ubuntu 14.04 (Linux kernel 3.14)
- Artesyn MVME2500, running Flop (Linux kernel 4.7)
- Jetway NUC, running Flop (Linux kernel 4.7)
- Beaglebone White and Black, running Flop (Linux kernel 3.14)
- Beaglebone AI, running Voltumna (Linux kernel 5.4)
- Altera Sockit, running Voltumna (Linux kernel 5.4)
- Supermicro Intel-based rack-mount servers, running Voltumna (Linux kernel 5.4)
- Dell Intel-based rack-mount servers, running Voltumna (Linux kernel 5.4)

Moreover, each accelerator deserves a central control system cluster, based on Proxmox, running a large number of virtual machines, where all the standard network services, as well as the Tango databases and the high-layer Tango devices run. All these virtual machines run Ubuntu 18.04 LTS. Control room workstations currently run Ubuntu 18.04 LTS as well.

As a side note, Voltumna is also available as virtual machine image, providing an effective approach to a totally reproducible deployment for virtual machine hosts, based on revision-control.

DEVELOPMENT ENVIRONMENT

Depending on the target system, different approaches are used to build the applications. For legacy VME based systems, as well as older intel-based servers, native target compilation/build, based on reference development hosts, is in place. Newer systems, based on FLOP [1] and, more recently, on Voltumna Linux, exploit a dedicated cross compilation environment.

Since 2019 Git is used for revision control of all software components developed in-house. For control systems software, a structure has been defined to easily navigate repositories, and conventions are in place to guarantee a strict consistency within Git repositories.

DEPLOYMENT REQUIREMENTS

Control system integrity and robustness requirements prevent from spreading administrator access rights to all the developers. Instead, system administrators take care of maintaining and servicing the control systems hosts. However, developers want/need flexibility, and not to depend on sysadmin for installing or updating applications and/or specific application configuration files. On the other side, there is the requirement to keep trace of who installed, updated or downgraded what, when and where. This requirement can be easily solved using modern CI/CD tools on reasonably recent hosts, but is unfeasible with old systems. To make this available on legacy platforms a custom build-and-deploy tool has been developed, named INAU which stands for AUtomatic INstallation, in Italian.

* lorenzo.pivetta@elettra.eu
† alessio.bogani@elettra.eu

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP1

INAU

INAU is based on two main services: the first, named "builder" takes care of building software components as soon as it's required, the second, named "installer" is in charge of interacting with developers (users) and installing upon request. INAU services share a MySQL database, to store all the configurations as well as the build and installation history, and a dedicated filesystem to keep all the build products.

INAU is mostly written in Python 3, with some small parts based on shell scripts to interact with legacy build hosts that do not provide Python or where the supported Python version is too old. INAU exploits the Python standard library, in particular multiprocessing but also some additional libraries, notably Flask RESTful [2] with SQLAlchemy [3], Python-ldap [4], Paramiko [5], and GitPython [6].

Currently 13 `tables` have been defined and are in use in the SQL back-end:

- *providers*: Git urls
- *repositories*: enabled repositories
- *distributions*: supported distributions
- *architectures*: supported architectures
- *platforms*: supported platforms
- *builders*: reference hosts for building
- *builds*: per-repository build information
- *artifacts*: per-build hash information
- *facilities*: list of deployment facilities
- *servers*: list of deploy servers
- *hosts*: list of deploy hosts
- *installations*: installation history
- *users*: enabled users

The `table` *providers* is used to keep the list of source code stores, e.g. Git servers; at Elettra that is the self hosted Gitlab server, community edition. The *repositories* `table` stores the list of Git repositories, enabled within INAU, where each repository is assigned a unique ID. Table 1 shows the Linux distributions configured in INAU, whilst the architectures available are *ppc*, *i686*, *x86_64* and *cascadelake-64*. Note that not all the combinations *distribution/architecture* are available, and the supported ones are stored in the `table` *platforms*.

Table 1: Supported Linux Distributions

id	name	version
1	Debian	3.0
2	Ubuntu	7.10
3	Ubuntu	10.04
4	Ubuntu	14.04
5	Ubuntu	16.04
6	Ubuntu	18.04
7	Ubuntu	10.04-caen
8	UbuntuDesktop	18.04
9	UbuntuDesktop	10.04
10	UbuntuDesktop	16.04
11	CCD	0.10beta22

The list of reference hosts available for building the software components is stored in `table` *builders*.

INAU design provides support for C++ applications, Python and Bash scripts and application-specific configuration files. Each "product", hereafter referenced as *artifact*, needs to be uniquely identified, and traceable, within the control systems: for each *artifact* INAU computes the hash[1] and stores it in the `table` *artifacts*.

INAU supports selective deployment, where the developer can specify where an artifact has to be installed. The concept of "facility", which at Elettra and FERMI mostly corresponds to the Tango facilities[2] in use, has been introduced. When installing, the usual case is to specify the target facility, which make the `artifact` available to all the hosts in the facility. As a special case, a single host can be specified.

NFS is used to provide shared storage between hosts in Elettra and FERMI control systems respectively, and each *facility* deserves at least one NFS server. The `table` *servers* is used to associate the *platforms* with the relevant NFS server and provide the installation prefix, that enables sharing different types of *artifacts* using the same NFS server.

The `table` *hosts* is used to keep track of the control system hosts. Each host is associated to the *facility* it belongs, the *server* providing the NFS storage and the *platform*.

The `table` *installations* records the installation requests made by the developers, keeping track of the target host, the build id, the type and the request date.

Moreover, in order to use INAU, the developer must be added to the list of authorized users.

For each Gitlab repository exploiting INAU, a web hook, that executes when a new annotated tag is pushed to the repository, has been configured. The builder service, acting as HTTP server, waits for the Gitlab message. When received, the builder service checks-out the specific tag version for all the *platforms* enabled and remotely builds the software component on the reference hosts by means of ssh. For each platform, build output files are kept between builds; when triggered INAU incrementally builds only what required. This speeds up building components also on slower legacy platforms. Whenever successful, the build is stored into the dedicated location.

The INAU installer service is, as well, based on an HTTP server that exports a REST API. Using a simple HTTP client, such as curl, the user can interact with the service. The installer service also supports authentication, based on LDAP. INAU block diagram is shown in Fig. 1.

USING INAU

The user can perform some useful actions, such as get the list of Gitlab repositories enabled in INAU, the list of *hosts* belonging to a *facility* or request some installation history. Moreover, the user can ask the installation of a repository,

[1] Currently sha256 in used

[2] A Tango facility is identified by the Tango database and all the hosts which belong to

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP1

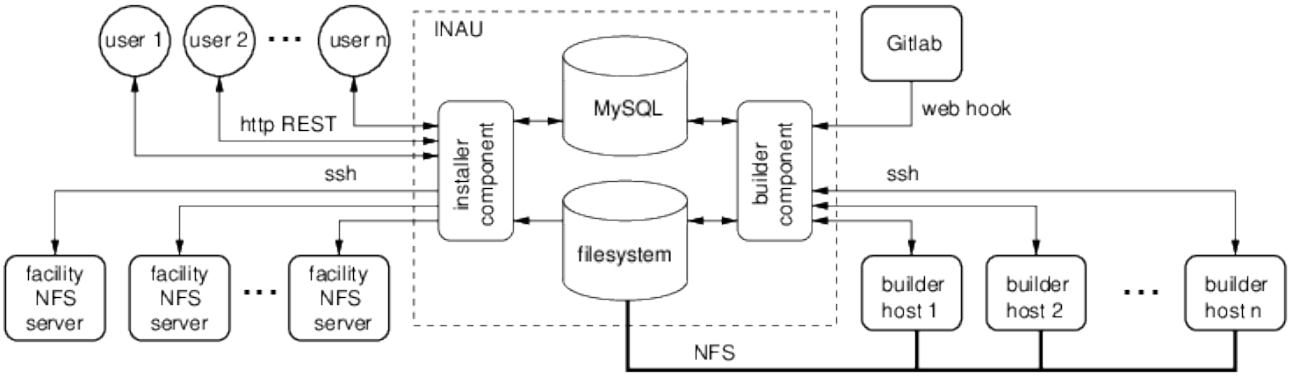


Figure 1: INAU block diagram.

Table 2: Interacting with INAU on the Command Line

```
$ curl https://inau.elettra.eu/v2/cs/repositories ⮐
id   name           provider             distribution  version arch   type      destination
-------------------------------------------------------------------------------------------
3    cs/ds/sfe      ssh://git@gitlab...  Ubuntu        7.10    ppc    cplusplus /bin/
...
1145 cs/gui/miotest ssh://git@gitlab...  UbuntuDesktop 18.04   x86_64 cplusplus /bin/
1148 cs/ds/hipace   ssh://git@gitlab...  Ubuntu        18.04   x86_64 cplusplus /bin/


$ curl https://inau.elettra.eu/v2/cs/facilities/elettra/installations ⮐
host               repository  tag    date                              author
-------------------------------------------------------------------------------
pcl-elettra-cre-01 cs/gui/acdc 1.0.3  Thu, 08 Sep 2022 14:43:39 -0000 lorenzo.pivetta
...
srv-ds-sre-02      cs/ds/4uhv  3.0.10 Wed, 07 Sep 2022 16:27:48 -0000 alessio.bogani


$ curl https://inau.elettra.eu/v2/cs/facilities/padres/installations
-u alessio.bogani -d"repository=cs/ds/4uhv" -d"tag=3.0.8" ⮐
```

specifying the facility, the annotated tag and the username that will be used for authentication.

Some INAU commands and their respective output are listed in Table 2.

CONCLUSION

INAU service is up-and-running since more than three years now, providing developers the autonomy and the flexibility they require, but also guaranteeing a safe approach for system integrity. INAU allows to maintain an accurate and always up-to-date view of applications and application-specific configurations deployed in legacy production systems.

A custom CI/CD service also brings in, as a marginal benefit, a total independence from commercial CI/CD tools that, from time to time, use to change the service policy or introduce proprietary code/plugins.

ACKNOWLEDGEMENTS

Thanks to the Control Systems group members and to the Software for Experiments group members for the useful discussions and suggestions.

REFERENCES

[1] L. Pivetta *et al.*, "FLOP: customizing Yocto project for MVME5xxx PoperPC and Beaglebone ARM", in *Proc. ICALEPCS'15*, Melbourne, Australia, Oct. 2015, pp. 958–961. doi:10.18429/JACoW-ICALEPCS2015-WEPGF112

[2] Flask RESTful, https://flask-restful.readthedocs.io/en/latest/

[3] SQLAlchemy, https://www.sqlalchemy.org/

[4] Python-ldap, https://www.python-ldap.org/en/python-ldap-3.4.2/

[5] Paramiko, https://www.paramiko.org/

[6] GitPython, https://gitpython.readthedocs.io/en/stable/

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP2

EPICS MODULE FOR BECKHOFF ADS PROTOCOL

Jernej Varlec*, Jure Varlec, Žiga Oven, Cosylab d.d., Ljubljana, Slovenia

Abstract

With increasing popularity of Beckhoff devices in scientific projects, there is a rising need for their devices to be integrated into EPICS control systems. Our customers often want to use Beckhoff PLCs for applications that must handle many inputs with fast cycle times. How can we connect Beckhoff devices to EPICS control systems without sacrificing this performance?

Beckhoff offers multiple possibilities when it comes to interfacing with their PLCs or industrial PCs, such as Modbus, OPC UA, or ADS protocol. While all of these could be used for the usual use cases, we believe that for more data intensive applications, ADS works best. For this reason, Cosylab developed an EPICS device support module that implements advanced ADS features, such as ADS sum commands, which provide fast read/write capabilities to your IOCs.

INTRODUCTION

When designing EPICS device support, there are two questions one usually considers: how the device support will communicate with the target devices, and which representations of data must be supported.

ADS for Communication

Invented by Beckhoff, *Automation Device Specification* (ADS) [1] is an open protocol that is used for interconnecting various Twincat software modules the company provides, such as event logger, HMI framework, or Twincat PLC runtimes, among others. These modules are considered as being independent virtual devices and form a server/client relationship; an example of this is a field on the HMI screen getting an update from a Twincat Runtime, all via ADS.

Within this relationship, as shown in Fig. 1, servers and clients communicate with ADS request/response messages which need to be, somehow, routed to the correct ADS device. This message routing is the responsibility of the AMS router, which is part of every Beckhoff Twincat device. The router checks the AMS header part of the message and reads the target port and address from it. Beckhoff provides fixed specification, called AMS ports [2], for their internal software modules. For example, typical Twincat 3 runtime uses AMS port 851.

The AMS ports are the first identifier required for communication within the Twincat system, the second part are the *AMS NetIDs*. These look similar to IP addresses with additional octets appended, and often they are: Users may find it easiest to just use their device's IP address, and then append "`.1.1`"[1] to it. Note that the AMS NetID can be anything, as long as it is unique.

In general, there are two types of ADS communication, asynchronous and notification. Asynchronous is exactly what it says on the tin: the client sends the request message to the server, then continues to operate normally until the server provides the client with the response, be it success or error. Notification-based communication, on the other hand, allows the clients to register themselves to the server, which then autonomously provides updates when values client registered change.

Beckhoff provides the ADS client functions in a C++ library published on Github [3].

IEC 61131-3 Data Types

Another thing to consider is what data types are used by a Twincat runtime. Twincat supports all the typical data types one would expect, such as signed an unsigned integer types up to 64-bit width, 32- and 64-bit floating point numbers, support for arrays and strings, pointer and reference types, and structures. But, since it conforms to the IEC 61131-3 standard [4], it also supports some more 'exotic' data types, such as 32-bit *DATE*, *DATE_AND_TIME*, *TIME_OF_DAY*, and their 64-bit versions, and a generic type, called *ANY*.

The list of supported types is quite long and supporting them all could prove to be a challenge. Another thing we must consider is the data types that EPICS records support. For example, *longin* record supports 64-bit signed integers, which means support for writing 64-bit unsigned types into this record type could prove to be problematic.

Of course, speed and stability should be considered as well. One might not think about it much as long their use case requires mere hundreds of reads or writes at a time,

[1] `.1.1.` is used here as a typical example

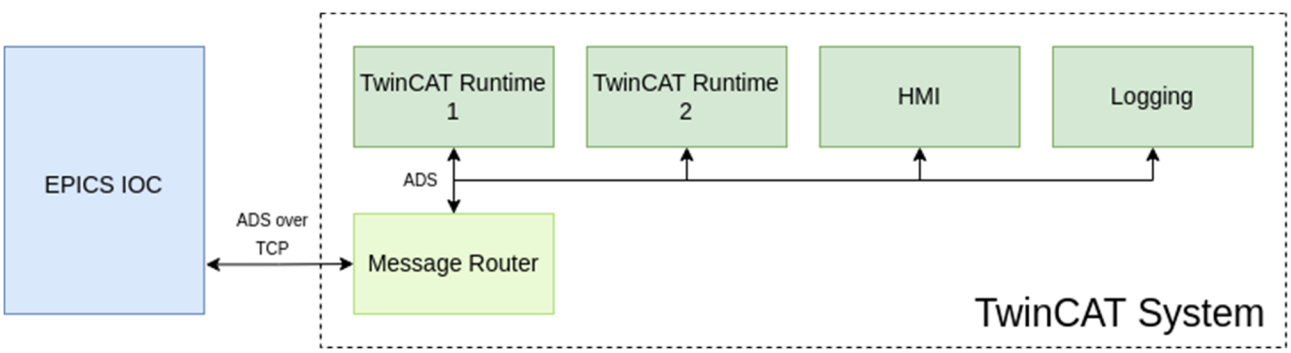


Figure 1: ADS communication via the Message router.

* jernej.varlec@cosylab.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP2

but the problem quickly escalates if the number of simultaneous reads jumps into four digits; the PLC has to dispatch a large number of responses, the message router must distribute them back to the client, and let's not forget the messages are distributed using TCP/IP. The overheads combined can result in severe performance impact, which can, in turn, result in EPICS scan threads being over-run, which is not desirable to say the least. And, it turns out, that speed and stability can go hand in hand.

ADS DEVICE SUPPORT

Autoparam

We solve the problem of supporting a large amount of data types with AutoparamDriver [5]. This adds a layer of abstraction above Asyn [6], which simplifies the device support a bit: one no longer has to write read and write functions for each Asyn interface. Instead, read and write function are defined for data types. The ADS device support only needs to implement the read and write functions for:

- Integers
- Digital I/O
- Floating point numbers
- Arrays

Data types surrounding time and date are not supported directly. Inside a Twincat system, these are represented by *DWORD* 32-bit unsigned integer type. There is a possibility of this being implemented at some point when a use case is identified.

Unsigned 64-bit integers, e.g., *LWORD* or *ULINT*, are also not supported. EPICS records, that would be likely targets for those types, only support 64-bit signed types.

Sum Commands

ADS C++ library developed by Beckhoff provides standard read and write methods, which are fine as long as there aren't too many requests sent at once. Because the ability to read thousands of variables and being stable is a requirement for the ADS device support, plain read and write methods are not good enough. Thankfully, ADS provides another option named 'Sum commands' [7], which allow us to pack many variables into a single ADS message. This is similar to what the EPICS Modbus module does, except it is not limited to 125 16-bit registers. It can theoretically read any data type and does not have an upper limit to how many variables in a chunk it can read. Practically, there are two limitations:

1. AMS message router can only handle 2048kB of data in a single message.
2. PLC cannot start another cycle until it resolves all ADS requests. This means that the PLC CPU can be stalled if one requests a lot of variables in a single chunk.

For the above reason, Beckhoff recommends no more than 500 variables per read, and this is also the default for the ADS device support. If the EPICS developer knows that they can afford slower PLC cycle times, they can increase this number at IOC init.

ADS device support implements sum reads as a default reading option; the device support automatically organizes all requests from records into chunks, and starts a scan thread, which continuously retrieves fresh values from the target PLC. Sum writes are not supported at the moment.

Using the Device Support

In order to use ADS device support, EPICS integrator needs to know how to interface through EPICS records and how to initialize the device support through EPICS start-up script.

EPICS interface, what we could also call *Address format*, or Asyn *reason*, is comprised of the:

1. **Data type:**
   specifies one of the supported PLC data types, e.g., *USINT, LREAL, BOOL*, etc. If the target variable is an array, append the '[]' to the datatype, except for strings, e.g., *USINT[], LREAL[], STRING*.
2. **Number of element (if requesting arrays):**
   is used to specify number of elements for array access, as well as to specify the length of *STRING* PLC variables, e.g., N=25.
3. **Operation (ADS command):**
   specifies if the PLC variable is read (R) or written (W).
4. **AMS port:**
   port in string or numerical format, e.g., *P=PLC_TC3*
5. **ADS variable:**
   ADS variable name in string format, e.g., *V=Main.temperature.*

An example, in which one wishes to read a *BOOL* variable named *switchStatus*, would be: *field(INP, "@asyn(test 0 0)* ***BOOL R P=PLC_TC3 V=switchStatus****")*

CONCLUSION

When testing, we wanted to measure the time between each ADS read and if scan threads are being over-run. We were testing the ADS device support in the following setting:

- PLC and IOC were in the same network.
- PLC cycle time was set to 10 ms.
- PLC variables being transferred were 64-bit *LREAL*.
- Number of variables was between 1000 and 10000.
- Around 500 samples were gathered for each SCAN rate.

As shown in Fig. 2, the device support proved to be stable at all SCAN rates (tested from .1 to 1 second) for any number of 64-bit variables up to 10000. Some more time was required for *I/O Intr* scan rate, but that appears to be due to comparing new value with the old one in order to detect value change.

REFERENCES

[1] Beckhoff ADS, https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/11291871243.html&id=6446904803799887467

[2] AMS Ports, https://infosys.beckhoff.com/english.php?content=../content/1033/tcinfosys3/11291871243.html&id=6446904803799887467

[3] Beckhoff ADS C++ Library, https://github.com/Beckhoff/ADS

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP2

Number of records	Record scan rate	Average [s]	Minimum [s]	Maximum [s]	Expected [s]	Number of records	Record scan rate	Average [s]	Minimum [s]	Maximum [s]	Expected [s]
1000	.1 second	0,100	0,100	0,100	0,100	5000	.1 second	0,100	0,100	0,100	0,100
1000	.2 second	0,200	0,200	0,200	0,200	5000	.2 second	0,200	0,200	0,200	0,200
1000	1 second	1,000	1,000	1,000	1,000	5000	1 second	1,000	1,000	1,000	1,000
1000	I/O Intr	0,010	0,006	0,020	nil	5000	I/O Intr	0,326	0,302	0,372	nil
2500	.1 second	0,100	0,100	0,100	0,100	10000	.1 second	0,100	0,100	0,100	0,100
2500	.2 second	0,200	0,200	0,200	0,200	10000	.2 second	0,200	0,200	0,200	0,200
2500	1 second	1,000	1,000	1,000	1,000	10000	1 second	1,000	0,991	1,013	1,000
2500	I/O Intr	0,050	0,044	0,071	nil	10000	I/O Intr	1,362	1,282	1,443	nil

Figure 2: ADS device support test results.

[4] IEC 61131-3, https://webstore.iec.ch/publication/4552

[5] AutoparamDriver, https://epics.cosylab.com/documentation/autoparamDriver/

[6] Asyn driver, https://epics-modules.github.io/master/asyn/

[7] ADS sum commands, https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_adsdll2/124835083.html&id=

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP3

STORAGE RING MODE FOR FAIR

Andreas Schaller, Jutta Fitzek, Hanno Christian Hüther, Raphael Müller,
Benjamin Peter, Anneke Walter, GSI, Darmstadt, Germany

Abstract

For the future Facility for Antiproton and Ion Research (FAIR), which is currently under construction, a new Control System is being developed and already used at major parts of the GSI facility. The central component for Settings Management within the FAIR Control System is based on CERN's framework "LHC Software Architecture" (LSA) and enhanced by FAIR specific features. One of the most complex features is the control mechanism of storage ring operations, the so-called Storage Ring Mode. This operation mode allows to manipulate device settings while the beam is circulating in the ring. There are four different types of possible changes in the Storage Ring Mode: skipping, repetition, breakpoint and manipulation. The Storage Ring Mode was developed in late 2019 and first used with beam in 2020 at the existing heavy ion Storage Ring ESR at GSI. This contribution illustrates in detail how the Storage Ring Mode is implemented within LSA and other subsystems. It also shows how it is operated using the Expert Storage Ring Mode application.

MOTIVATION

To reliably ensure deterministic behavior, the new Timing System for FAIR executes predefined event sequences. The first use case realized in the FAIR Control System was synchrotron operation, based on fully pre-planned schedules, which has been successfully used since 2015.

In 2019, the FAIR Control System was enhanced to support storage ring operation [1]. The old control system supported storage ring operation at GSI since 1990, with beams often being stored for several hours and up to a week. The Storage Ring Mode of the new control system now also supports the required flexibility and allows interactive schedule changes, including in-cycle modifications of stored beams.

The basics of Storage Ring Mode were presented at ICALEPCS in 2021 [1]. This contribution provides more insights into the realization of its features.

STORAGE RING MODE - COUPLING WITH SOURCE MACHINE

Storage Ring Mode allows for interactive, dynamic operation and very long, arbitrary beam storage times. Therefore, event sequences in the Timing System for storage rings are usually completely independent and separated from the event sequences of other accelerators.

However, when transferring beam from a source accelerator into a storage ring, the event sequences of both machines have to be synchronized. This is realized by a so-called coupling mechanism. The FAIR Control System currently supports two types of coupling: strong and weak coupling.

Strong Coupling

Strong coupling guarantees that the beam is properly injected by having the storage ring explicitly request beam and wait until it is ready to be injected.

As shown in the left part of Fig. 1, the storage ring prepares for beam injection, requests beam from the injector, and finally enters a wait loop. The injector starts creating the beam once the overall schedule allows it. When beam is ready, the injector sends a command to the storage ring to leave the wait loop. Extraction from the injector and injection into the storage ring then continue synchronously.

This type of coupling is used by the ESR Heavy Ion Storage Ring when receiving beam from the SIS18 ring.

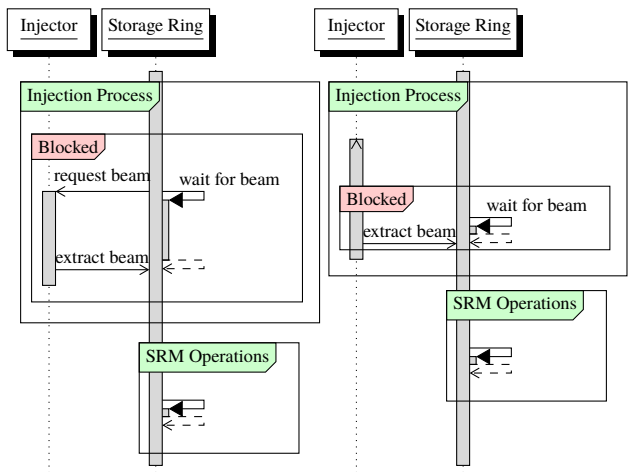


Figure 1: Sequence diagram of a strong coupling (left) and weak coupling (right).

Weak Coupling (Fire and Forget)

With weak coupling, the injector *always* produces and extracts beam without making sure that the storage ring is ready for transfer ("fire and forget"). This coupling mode is used to allow the storage ring to serve other experiments while the beam is produced, instead of just waiting.

As shown on the right part of Fig. 1, the injector produces beam without an explicit beam request. When beam is ready, the injector notifies the storage ring to leave the wait loop. At this point, the storage ring must be ready for injection when the injector starts extraction. Operators achieve this by carefully aligning the schedules of the injector and the storage ring. If the storage ring is not ready for beam at this point, the notification is ignored and beam is lost.

This mode is used at CRYRING@ESR [2], enabling CRYRING to accept beams from its local linear accelerator in synchrotron mode while waiting for ESR's beam.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP3

STORAGE RING MODE – FEATURES

Storage Ring Mode introduces smaller building blocks within a Chain [3] that represents the full storage ring cycle. These building blocks are essential for the realization of the four key features: breakpoints, repetitions, skipping, and manipulations. All features can be combined with each other within a building block, except for manipulation. Boolean signals, which are handled by the Beam Scheduling System (BSS) [4], are used by the operators to interactively enable and disable these features.

In the following, each of the features will be described by looking at their corresponding timing graph representation.

Breakpoint

Breakpoints allow to interactively pause the schedule execution at the end of a building block while the beam is still circulating in the ring [1]. As shown in Fig. 2, the timing graph contains an event loop that is repeatedly executed as long as the associated signal is true.

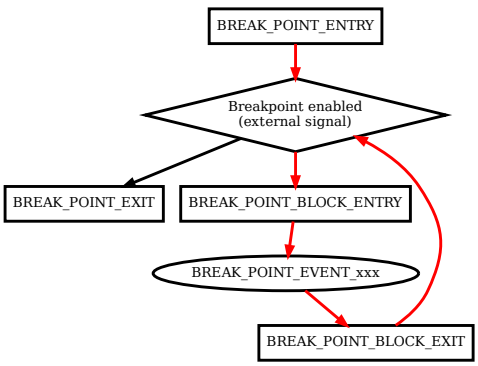


Figure 2: Timing Graph: Breakpoint.

Repetition

Blocks of the Storage Ring Chain can be repeated for a configurable number of times [1]. This is realized as a loop with a counter in the timing graph, as shown in Fig. 3. The counter gets decreased by the timing system each time the loop is entered. Operators can abort currently executed repetitions with an explicit command to the BSS, which immediately sets the counter to zero, thus preventing further repetitions.

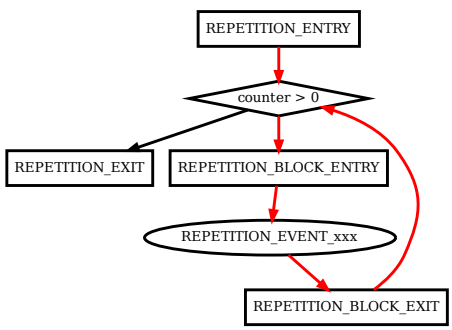


Figure 3: Timing Graph: Repetition.

Skipping

Blocks of a Storage Ring Chain can be skipped based on the state of the associated signal, which is set by operators [1]. The timing graph therefore contains a direct path, bypassing the blocks to be skipped. This is visualized in Fig. 4.

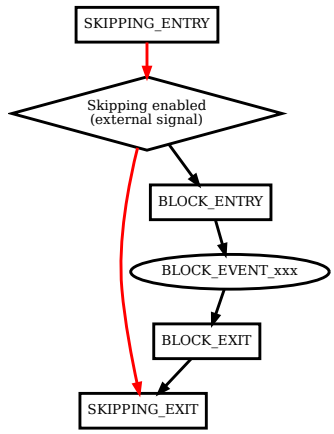


Figure 4: Timing Graph: Skipping.

Manipulation

The most complex Storage Ring Mode feature is the manipulation. It offers the possibility to pause execution and repeatedly modify certain settings at pre-defined points in the Storage Ring Chain. The devices are then ramped to the new values once, influencing the beam while it circulates. The changes are incorporated into the surrounding settings for the next Chain execution [1]. In the timing graph shown in Fig. 5, the red path is where setting modifications are performed, while the blue path describes the pause loop.

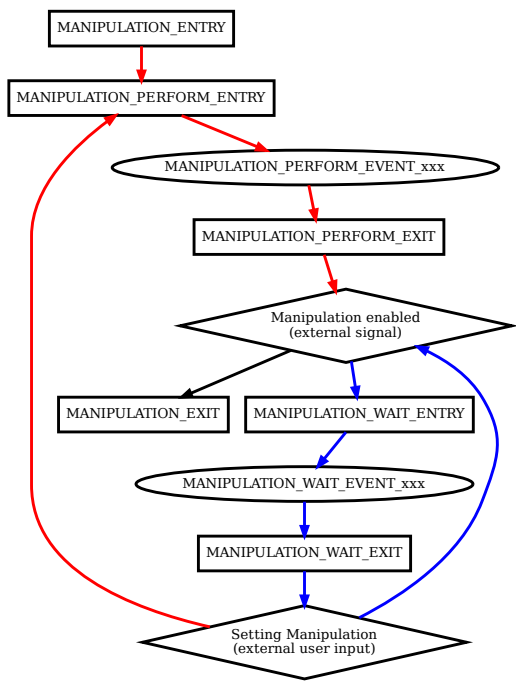
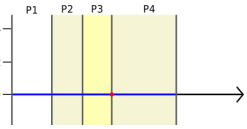


Figure 5: Timing Graph: Manipulation.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP3

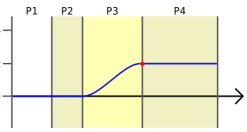
Figure 6 illustrates how the manipulation works from a settings-oriented view: A setting is changed to a new value, resulting in a small ramp that is executed exactly once. The new value is incorporated into the surrounding settings.

Initial Situation

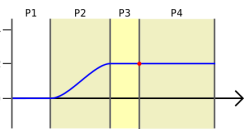


Beam execution pauses by looping P3. Value is currently 0.

First Manipulation

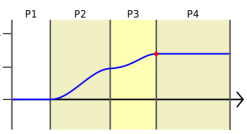


User changes endpoint of P3 from 0 → 2, manipulation ramp gets calculated, sent to devices and executed once. Value is now 2.

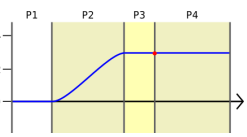


Value was sent, flatten P3 and merge ramp into P2. This preserves the value change for the next beam execution.

Second Manipulation

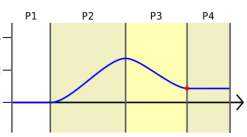


User changes endpoint of P3 from 2 → 3, manipulation ramp gets calculated, sent to devices and executed once. Value is now 3.

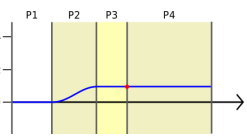


Value was sent, flatten P3 and merge ramp into P2. The result is the direct ramp from start value of P2 to the new start value of P3.

Third Manipulation



User changes endpoint of P3 from 3 → 1, manipulation ramp gets calculated, sent to devices and executed once. Value is now 1.



Value was sent, flatten P3 and merge ramp into P2.

Figure 6: Setting Manipulation (exemplary).

STORAGE RING MODE APPLICATION

All Storage Ring Mode features can be configured and controlled by operators using the specialized Storage Ring Mode Application (StoRiMo).

StoRiMo provides an overview of the different building blocks of a Storage Ring Chain, see Fig. 7. The red border indicates the current point of execution, which is determined by observing executed timing events. Small indicator icons visualize the features available at a Storage Ring Mode block. Signals are displayed below and can be switched by operators to enable or disable these features.

Figure 7: Partial screenshot of StoRiMo.

INVOLVED SUBSYSTEMS

StoRiMo is the operators' interface for controlling the Storage Ring Mode. It monitors and controls Storage Ring Mode features using interfaces of LSA and BSS. LSA calculates device settings, coordinates the manipulation operation and generates the timing graph. This graph is then provided to BSS and the Timing Master, a White Rabbit-based timing system which executes the pre-defined events. BSS reacts to signal changes from StoRiMo and updates the Timing Master's timing graph to reflect which paths of the graph shall be executed.

SUMMARY AND OUTLOOK

Storage Ring Mode and the specialized StoRiMo application were developed for the new Control System for FAIR at GSI. They were successfully used in production during several beamtimes at ESR and CRYRING. The flexibility provided by the new features allowed for multiple machine and science experiments.

A requested, but not yet implemented feature is the possibility of converting Storage Ring Mode contexts to Synchrotron Mode contexts (and back). This would allow for deterministic schedules once the beam is setup, e.g. for synchronizing with another machine.

REFERENCES

[1] R. Mueller *et al.*, "Supporting Flexible Runtime Control and Storage Ring Operation with the FAIR Settings Management System", presented at the ICALEPCS'21, Shanghai, China, Oct. 2021. doi:10.18429/JACoW-ICALEPCS2021-WEPV047

[2] F. Herfurth *et al.*, "Commissioning of the Low Energy Storage Ring Facility CRYRING@ESR", in *Proc. COOL'17*, Bonn, Germany, Sep. 2017, pp. 81–83. doi:10.18429/JACoW-COOL2017-THM13

[3] H. C. Hüther, J. Fitzek, R. Mueller, and D. Ondreka, "Progress and Challenges during the Development of the Settings Management System for FAIR", in *Proc. PCaPAC'14*, Karlsruhe, Germany, Oct. 2014, paper WPO005, pp. 40–42.

[4] S. Krepp, J. Fitzek, H. C. Hüther, R. Mueller, A. Schaller, and A. Walter, "A Dynamic Beam Scheduling System for the FAIR Accelerator Facility", presented at the ICALEPCS'21, Shanghai, China, Oct. 2021. doi:10.18429/JACoW-ICALEPCS2021-MOPV013

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP5

LASER PULSE DURATION OPTIMIZATION WITH NUMERICAL METHODS

Francesco Capuano[1,*], Davorin Peceli, Bedřic Rus, Alexandr Špaček
ELI, Institute of Physics, Czech Academy of Sciences, Dolní Brežany, Czech Republic
Gabriele Tiboni[†], Politecnico di Torino, Turin, Italy
[1]also Politecnico di Torino, Turin, Italy

Abstract

In this study we explore the optimization of laser pulse duration to obtain the shortest possible pulse.

We do this by employing a feedback loop between a pulse shaper and pulse duration measurements. We apply to this problem several iterative algorithms including gradient descent, Bayesian optimization and genetic algorithms, using a simulation of the actual laser represented via a semi-physical model of the laser based on the process of linear and nonlinear phase accumulation.

INTRODUCTION & RELATED WORKS

L1-Allegra System

L1 Allegra system is high power laser system developed in ELI Beamlines in Czech Republic designed to deliver <20fs pulses with energy higher than 100 mJ at a repetition rate of 1 kHz. This system is based on the amplification of frequency-chirped pico-second pulses in an Optical Parametric Chirped Pulse Amplification (OPCPA) chain consisting of seven amplifiers. L1 contains 7 OPCPA stages that are pumped by 5 diode pumped lasers based on commercial Ybdoped thin-disk Regenerative Amplifiers (RA) DIRA 200-1. All of the pump lasers generate pulses at 1030 nm with 1 kHz repetition rate and also include stretcher, compressor and an LBO crystals for second harmonic generation (SHG) of pulses at 515 nm. To achieve high SHG efficiency several features of the pump laser system should be optimized. One of these features is the laser pulse temporal shape. The optimization of temporal pulse shape is accomplished through the manipulation of the pulse spectral phase. A standard method for spectral phase representation employs a Taylor expansion of spectral phase, $\varphi(\omega)$, around the central angular frequency ω_0, of the spectral pulse:

$$\varphi(\omega) = \sum_{=0}^{\infty} \frac{\partial^{(\ )}\phi}{\partial\omega^{(\ )}} \cdot (\omega - \omega_0) \ . \tag{1}$$

The first two terms of the sum of Eq. (1) represent constant phase and group delay and don't have effect on the pulse temporal shape. Third, fourth and fifth term are instead related to the dispersion parameters group delay dispersion (GDD $\sim \varphi'$), third order dispersion (TOD $\sim \varphi''$) and fourth order dispersion (FOD $\sim \varphi'''$) and do have an impact on the temporal profile. In this work, we aim at control these parameters, to which we will collectively refer to as ψ.

* Francesco.Capuano@studenti.polito.it
† Gabriele.Tiboni@polito.it

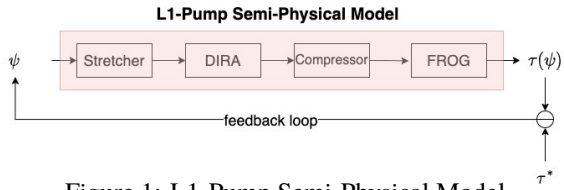


Figure 1: L1-Pump Semi-Physical Model.

Since SHG is an intensity-dependent non-linear optical process, the highest conversion efficiency is obtained for those pulses with temporal shape closest to transform-limited pulses. In particular, since the transform-limited (TL) pulse is the pulse with minimum possible pulse duration given the present spectral bandwidth, it is guaranteed to have the highest possible pulse intensity.

Temporal properties of amplified pulses are controlled through manipulation of parameters of the stretcher by *Teraxion* and monitored using SH FROG 1030 by *Femtoeasy*. The control of pulse temporal shape is done by manipulating three parameters, d_2, d_3 and d_4 on the stretcher controls that relate to parameters GDD, TOD and FOD previously introduced through a determinate system of linear equations, depending on the central wavelength (derived from central frequency) of the spectrum considered.

Related Works

Obtaining short and sharp laser pulses is crucial for running high-efficiency physical systems that would reach really high intensities with limited energy consumption.

Various algorithms were used in ultra-fast laser systems for active feedback optimization of laser parameters or laser-matter interactions [1, 2], but none of these previous approaches optimized the single pulse shape.

In this work we compare three different algorithms for the optimization of the temporal profile of laser pulses. These are: Bayesian Optimization, Differential Evolution, and a custom implementation of gradient descent.

Semi-Physical Model of L1-Pump-System

Optimization of laser pulse shape is designed through feedback control loop between stretcher and FROG. To test performances of different optimization algorithms we designed a theoretical semi-physical model of the L1 amplifier chain containing stretcher, DIRA, compressor and FROG, as presented in Fig. 1.

Each of the elements in the pump chain contribute to the overall spectral phase. Furthermore, while stretcher and compressor introduce only linear change to the spectral phase (GDD, TOD and FOD), DIRA also introduces nonlinear

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP5

effects. The system presented in Fig. 1 is parameterized by: **B-integral** (representing a measure of the accumulation of light's nonlinear phase), estimated to be 2 for our DIRA; **Compressor parameters**, considered constant in this work and estimated as opposite to the nominal values of stretcher's parameters.

We conclude the presentation of our semi-physical model highlighting how DIRA's contribution in terms of linear spectral phase can be neglected.

PULSE OPTIMIZATION

Other works did tackle the problem of laser optimization focusing on using Deep Reinforcement Learning to obtain mode-locking inside the laser cavity [2]. However, no previous work was done to optimize the actual pulse shape of single pulses so as to make it as similar as possible to an arbitrary target shape.

In this work (freely available on GitHub [3]), we focus exactly on this problem. If one indicates with τ^* such target shape and with $\tau(\psi)$ the temporal shape resulting from the adoption of the control ψ on the system, then it is possible to frame the problem as:

$$\min_{\psi \in \mathcal{X}} L\big(\tau^*, \tau(\psi)\big) \,, \tag{2}$$

where $L : \mathbb{R}^n \mapsto \mathbb{R}^+$ is a figure of merit which is used to guide the optimization route towards better and better values of ψ and $\mathcal{X}$ is the region of the parameters space containing physically-feasible configurations.

In this work we adopted as target temporal profile the transform-limited pulse, i.e. the temporal representation of the pulse corresponding to a zero phase imposed on the original spectrum.

In the case in which the system presented in Fig. 1 were not present any non-linear blocks such as regenerative amplifiers, this result can be obtained imposing a control which corresponds to $\psi_{str.} = -\psi_{compr.}$. However, since this nonlinearity is indeed present in the system (as the beam energy needs to be increased), it is not possible to obtain the transform-limited pulse applying such control.

This can also be seen in Fig. 2, where it is possible to see that the non linearity present heavily distorts the output of the application of an otherwise optimal control.

Loss Function

We shall first observe that an exact analytical solution to the problem of mixed (linear and non-linear) phase accumulation it is practically impossible to obtain in our case.

This fact generates the need of adopting a solution that, with an iterative approach, adjusts the control parameters so as to find a region in which the control induces a temporal profile matching the desired one under an acceptable threshold. Such a threshold is computed with respect to an arbitrary, yet very relevant, L.

Our first guess has been the Mean Squared Error (MSE) loss function. However, we observed that this loss function is really not informative as a guiding signal for the optimization process. This is due to two main reasons:

- Since the temporal profile is represented via a 0-1 normalization, differences are also in this range. This implies that differences in the order of magnitude of one tenth -which are very significant in this case- account only for values in the order of magnitude of one cent, when squared (according to MSE formulation). Since the majority of the two curves (the tails of the impulse) numerically assumes very similar values, the MSE's numerator is very likely to be a small number.
- The algorithm we used to perform Fourier Transformations requires a very large number of points (often 30k) in order to obtain temporal profiles with acceptable resolutions. This essentially means the MSE's denomiator would always be a large number, independently on the actual temporal shapes, thus biasing the final a value towards small values.

In light of this, we used different loss functions for our problem, as MSE would be inevitably biased towards small numbers and, therefore, not a valuable guiding signal in any optimization route.

Optimization Algorithms

To obtain the configuration of parameters which minimizes one of the losses considered, we implemented various optimization algorithms. In particular, we implemented three algorithms: two so-called "black box" algorithms, namely Bayesian Optimization and Differential Evolution, and one "first order" algorithm based on first-order differential information employed in an implementation of the ADAGRAD algorithm [4].

It is important to note that while both the black-box algorithms can, in principle, be applied to the real-world laser in the sake of optimizing its configuration, ADAGRAD cannot. ADAGRAD could be applied approximating if the gradients were to be approximated with finite differences, but this would very much increase the computational effort required.

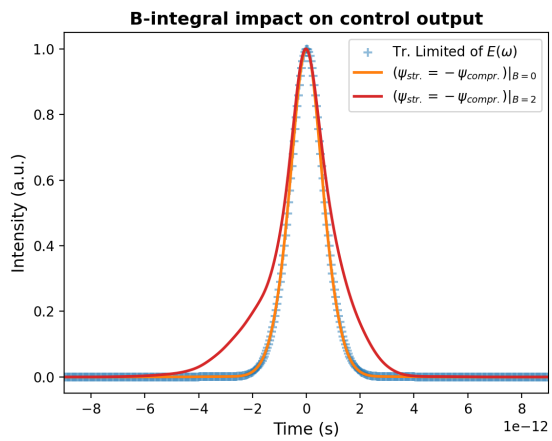


Figure 2: Impact of B-integral and control parameters on output shape.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP5

However, since we plan on collecting enough real data to train an Articial Neural Network (NN) to reconstruct the $\psi \mapsto \tau$ mapping (which would support automatic differentiation), we decided to experiment with this algorithm despite its immediate lack of practical applicability.

Bayesian Optimization Bayesian Optimization is a popular black-box optimization algorithm that probes at random various points inside a feasible region $\mathcal{X}$, adjusting, according to Bayesian Theorem, a probability distribution on the function fitting best those points.

This function is referred to as "surrogate function" or, more commonly, "acquisition function". At each iteration, the new point to be probed is selected balancing between exploration of the search space and maximization of this acquisition function.

We want to stress that this algorithm should not be directly applied on the real hardware, as the exploration of the search space causes serious oscillations between subsequent probed points, thus posing the actual dynamical system under a large amount of stress.

As a final note, it is important to note that in our implementation we restricted the number of iterations to 150, as we observed performance improvement stagnation over around 120 iterations and obtained a result presented in Fig. 3. As it is possible to see, the obtained results presents some wings that make it significantly different from the target profile for what concerns pulse mounting and de-mounting. Those wings could probably be by further exploring the GDD and TOD search space.

Among all the different loss functions tested, the one that gave the best results is the **sum-L1-Manhattan norm**. We used the default acquisition function-exploration coefficient (κ) configuration of [5] as hyper-parameters.

Differential Evolution Like Bayesian Optimization, Differential Evolution (DE) is a black-box algorithm that explores a unknown landscape in a feasible region $\mathcal{X}$.

This algorithm mimics the process of natural selection, initializing a population of candidate solutions that iteratevely gives birth to mixed individuals.

The fitness of the various candidate solutions is typically strictly related to the value of the objective function evaluated at each individual point. The algorithm is defined so as to promote some characteristics of the best individual in each generation's offspring, so as to have that as generation go by, the fitness level increases as well as the quality of the candidate points.

While with this algorithm we observed a significant decrease in oscillations between subsequent points (particularly at the beginning), we must observe that such oscillations are most likely not very dangerous for the actual system, since they are very rare.

However, this solution may be really slow when applied to the actual machinery. Such a increase in the needed time is mainly due to the fact that, when in the real world, each function evaluation lasts at least as the actual physical process underlying the control application. Since DE is a method which requires a high number of function evaluations, it is clear how this method could turn out to be really slow as minutes may be necessary to complete only one of a multitude of necessary iterations, especially if starting in a region that eventually turns out to be far apart from the optimal point.

Among all the different loss functions tested, the one that gave the best results is a mixture of one which computes the **MSE only for non-zero values**, (thus discarding the pulses' long tails) and a measure of **the mismatch in terms of subtended area by controlled and target pulse** (weighted in favour of difference of area, respectively with 0.3-0.7 weights).

As a final note, it is important to note that in our implementation we restricted the number of iterations to 150 and the population size to 20. The mutation coefficient, responsible to combine population's individuals has been set to 0.8 whereas the recombination probability between the best candidate and the mutant has been set to 0.7. The results obtained with this algorithm turned out to be far better than the ones obtained with Bayesian Optimization and are presented in Fig. 3. It is possible to see a pedestal which indicates a sub-optimal exploration of the TOD and FOD space, but the obtained pulse is very much similar to the transform limited for the considered set of parameters.

ADAGRAD Unlike the two other proposed approaches, ADAGRAD assumes the possibility of accessing first-order differential information so as to build an optimization algorithm which ultimately differs from regular steepest descent for its capability of adapting and scaling its step-length with respect to the dimension it is optimizing.

This is particularly interesting for our setting, as the feasible region $\mathcal{X}$ presents some peculiarities, such as the fact that between the first and the last dimension there are almost than 30 orders of magnitude. Therefore, an adaptive strategy for the learning rate must be put in place.

ADAGRAD was born as an unconstrained optimization algorithm, and has had a wide adoption in NN-training. To decline this algorithm to this constrained case we employed the-

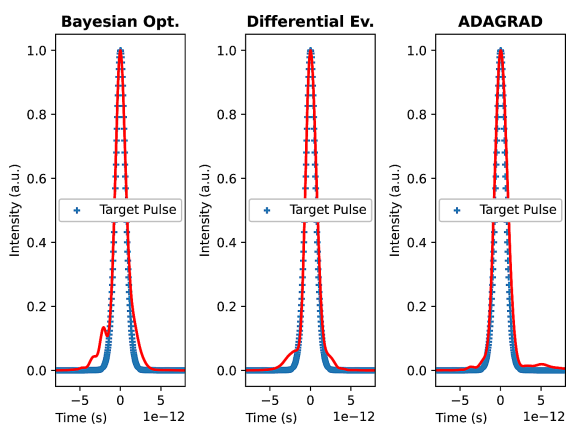


Figure 3: Final results obtained with different algorithms.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP5

Table 1: Final Results Obtained

Algorithm	Number of func. evaluations	Final MSE reached
Bayesian Opt.	150	2.625×10^{-5}
Differential Ev.	200 (its) $\times$ 20 (pop-size) = 4000	1.450×10^{-5}
ADAGRAD	3000 (its) = 3000	4.899×10^{-6}

oretical results presented in [6], such as Sequential Penalty Methods, turning the constrained optimization problem into a constrained one.

At each iteration, the optimization route is directed in the steepest descent direction using an adaptive learning rate which re-scales an initial learning rate η_0 using the inner product of all past gradients so far. This helps scaling the step-length using information on past steepest directions which, ultimately, leads to an adaptive learning rate. If the new point is outside of the feasible region, then the objective function is penalized with an increasing penalty terms that eventually pushes the optimization route into $\mathcal{X}$.

This algorithm is definitely the safest in terms of relative dissimilarity between two different candidate points (since this difference can be controlled using η_0) and the one requiring the smallest number of function evaluations. However, it is practically deployable only in the case in which a NN able to approximate the mapping $\psi \mapsto \tau$ is built. Alternatively, this method could be built using finite differences, thus resulting in eliminating the need for the ANN. In this case, the number of function evaluations could be contained to be even one order of magnitude smaller than the number of function evaluations needed in DE, for instance.

Among all the different loss functions tested, the one that gave the best results is the **natural logarithm of the sum-L1-Manhattan norm**. We used an exponentially increasing penalty term equal to $\frac{1}{10\cdot(0.9999)^k}$ and an initial learning rate of $\eta_0 = 10$. Moreover, we mapped ψ to its $\mathrm{fs}^2, \mathrm{fs}^3$ and fs^4 representation to improve the numerical stability of the algorithm.

The results obtained with this algorithm are presented in Fig. 3. As it is possible to see, ADAGRAD practically reproduces the transform-limited pulse, although the right shoulder of the pulse is a bit off. This is most definitely due to an exploration of the GDD dimension, which we plan on further analysing soon.

Results

The algorithms just presented have been all used to optimize the phase so as to obtain the most similar shape to transform-limited. It is worth noting that, while we did benchmark our model and observed that it captures some fundamental trend in actual FROG-reconstructed pulse shapes, we did limit the complexity of the model to keep its run-time in bearable time for algorithms employing a typically large number of iterations.

Our results are presented in terms of obtained shapes in Fig. 3 and, for what concerns the final MSE reached, in Table 1.

CONCLUSION

In this work we showed how to effectively use different algorithms to modify the temporal shape of the L1-Allegra high-power laser pulse. The modification of the temporal profile has a wide range of applications, as it is a first step towards to reach the high intensities that users and researchers at ELI Beamlines demand. To obtain such result, we did explore the space of parameters with three different supervised algorithms, as they all needed a target temporal profile to carry out the parameters optimization.

However, we did observe a high dependency of the results found (respectively, the best configuration of parameters obtained with the different algorithms) on the parametrization of laser environment. This is a crucial problem, as precise knowledge about the laser's parametrization is typically non-obtainable.

As future work, we plan on expanding the Pulse Optimization process with techniques which are robust to such a problem. In particular, sequential decision making approaches -such as in Reinforcement Learning- may be investigated to achieve adaptive, online control of high-power lasers that further complies with real hardware constraints on machine safety.

REFERENCES

[1] T. Baumert, T. Brixner, V. Seyfried, M. Strehle, and G. Gerber, "Femtosecond pulse shaping by an evolutionary algorithm with feedback," *Appl. Phys. B: Lasers Opt.*, vol. 65, no. 6, 1997. doi:10.1007/s003400050346

[2] E. Kuprikov, A. Kokhanovskiy, K. Serebrennikov, and S. Turitsyn, "Deep reinforcement learning for self-tuning laser source of dissipative solitons," *Sci. Rep.*, vol. 12, no. 1, pp. 1–9, 2022. doi:10.1038/s41598-022-11274-w

[3] *Eliopt*. https://www.github.com/fracapuano/ELIopt

[4] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, no. 7, pp. 2121–2159, 2011. http://jmlr.org/papers/v12/duchi11a.html

[5] F. Nogueira, "Bayesian Optimization: Open source constrained global optimization tool for Python," 2014. https://github.com/fmfn/BayesianOptimization

[6] S. Wright, J. Nocedal, *et al.*, "Numerical optimization," *Springer Science*, vol. 35, no. 67-68, p. 7, 1999.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP6

MONOCHROMATOR CONTROLLER BASED ON ALBA ELECTROMETER Em#

A. Baucells †, G.Agostini, J.Avila-Abellan, C. Escudero, O.Matilla, X.Serra-Gallifa, O.Vallcorba
CELLS - ALBA Synchrotron, Cerdanyola del Vallès, Spain

Abstract

Guaranteeing that the X-Ray beam reaches the experimental station with optimal characteristics is a crucial task in a synchrotron beamline. One of the critical factors which can lead to beam degradation is the thermal drifts and the mechanical inertias present in the optical elements, such as a monochromator. This article shows a new functionality of the ALBA Electrometer (Em#), which ensures that the beamline receives the maximum possible beam intensity during the experiment. From the current reading of an ionization chamber and driving the piezo-actuator pitch of the monochromator, the Em# implements a Perturb and Observe (P&O) algorithm that detects the peak beam intensity while tracking it. This feature has been tested on NOTOS beamline and the preliminary results of the performance are shown in this paper.

INTRODUCTION

Monochromators are essential elements in a beamline to study the behaviour of materials and they actively play a role to perform spectra and scans in 3rd generation synchrotrons. While they are robust optical elements they are still subject to slight beam degradation caused by the mechanics inertia, temperature drifts and the parallelism of the crystals. To deal with this, several beamlines at ALBA have been using the Monochromator Controller from ESRF for many years. As the number of beamlines increases the need to guarantee stock and support of these controllers has led to explore the development of a monochromator controller using the ALBA Electrometer Em# [1]. Another motivation to implement this new functionality to the Em# is to use the monochromator controller in oscillation mode, allowing to operate at the maximum point of the intensity.

NEW FUNCTIONALITY OF THE EM#

The implementation of this new feature of the Em# is based on the usual arrangement of a beamline to monitor the intensity generated by an ion chamber in the end station. In particular, it has been designed considering the structure of the NOTOS beamline at ALBA. In the experimental hutch, an ion chamber is used at the sample environment generating a current that is measured by an Em#. The monochromator controller will apply an analogue voltage signal to the piezo amplifier that drives the piezo motor stacked over the pitch stage of the DMM monochromator in the Optical Hutch (Fig. 1). Due to this mechanical arrangement of the pitch crystal, the motion produced by both the stepper motor and the piezo will affect the same encoder reading on that stage. For that reason, the pitch axis has to operate in open loop to be closed then by the ion chamber, generating the current to be processed by the monochromator controller and applying a voltage to drive the piezo.

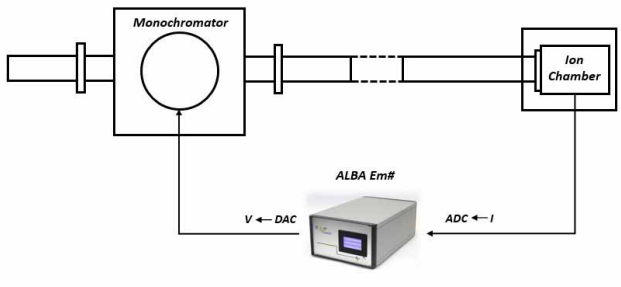


Figure 1: Schema of the Em# as monochromator controller.

The Em# will process the incoming values by executing a Maximum Power Point Tracking (MPPT) algorithm (Fig. 2), based on a Perturb and Observe (P&O) technique widely used for optimization of power management of solar cells [2].

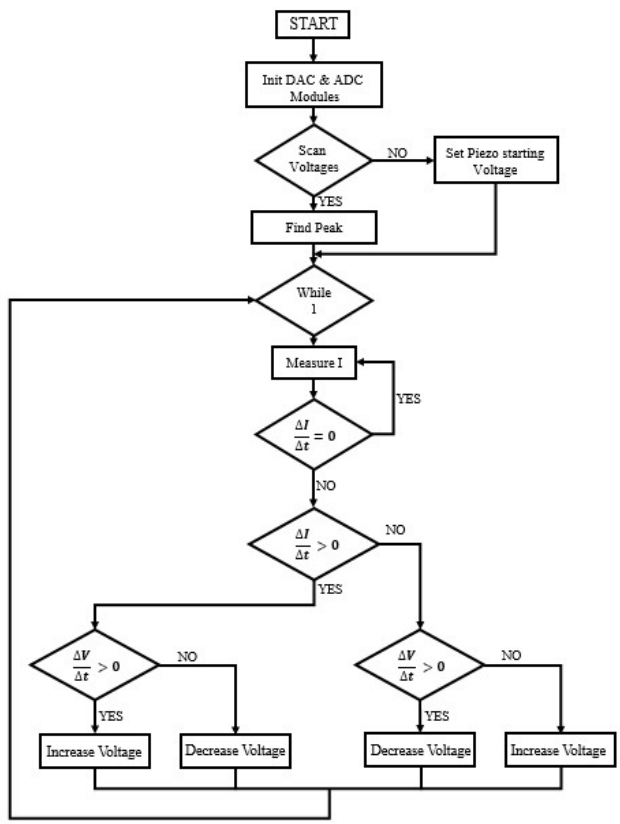


Figure 2: MPPT algorithm of the Em#.

† abaucells@cells.es

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP6

After initializing the modules for the acquisition, the Em# will do (if selected) a scan of the voltage range (0 to 10V) to find the peak of intensity and will set the piezo to the corresponding position. If the voltage scan is not performed, the user should set the starting voltage from which the controller regulates. At this point the Em# will constantly measure the current. After each measure, it will evaluate the current value with the past one. While the value is the same as the past one it will not change the voltage because is already on the maximum point. While the value is different, it will analyse if the current is increasing or decreasing and to which side of the peak is approaching by comparing the values of the voltages at each iteration.

PIEZO CHARACTERIZATION

Before starting the test of the monochromator controller operation, it was decided to test and characterize in open loop the behaviour of the piezo motor by reading the current generated in the ion chamber.

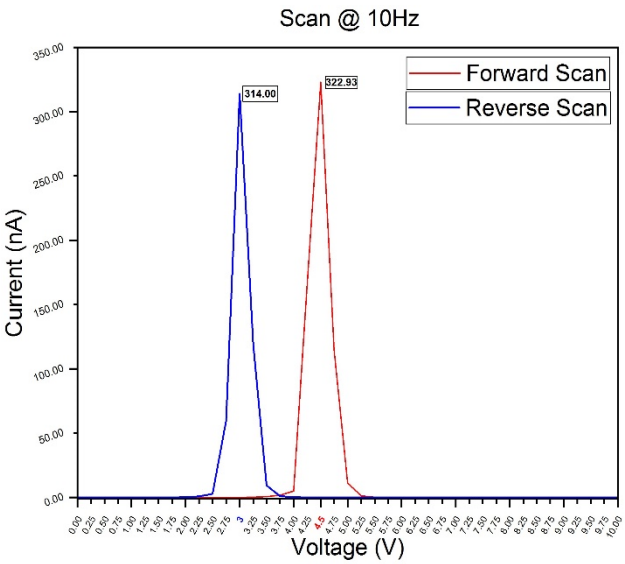


Figure 3: Scan of the voltages at 10Hz.

Figure 3 shows the curves of the scans of voltage to find the maximum value of the current (scan done before the while loop in Fig. 2). The frequency of this scan is 10Hz and the step is 0.25V. The red line belongs to the forward scan having the maximum of 322.93 nA corresponding to a piezo voltage of 4.5 V. In the reverse scan the peak value is 314 nA for a voltage of 3 V. It is clearly observed that the error occurred by the non-linearity issue of hysteresis is about 15% of the range.

As it is shown in Fig. 4 the peaks of the current curves (forward and reverse scans) are much closer. This scan has a step of 0.1 V and a frequency of 1 Hz. The error incurred by hysteresis is reduced to 0.2 V (2 % of error of the piezo travel range).

Comparing Figs. 3 and 4, it is seen how the hysteresis depends on the frequency of the voltage applied [3]. However it is questionable if this will entail a limitation in the controller operation because the delta steps applied to increase or decrease the piezo voltage are much lower than 0.1 V in real operation.

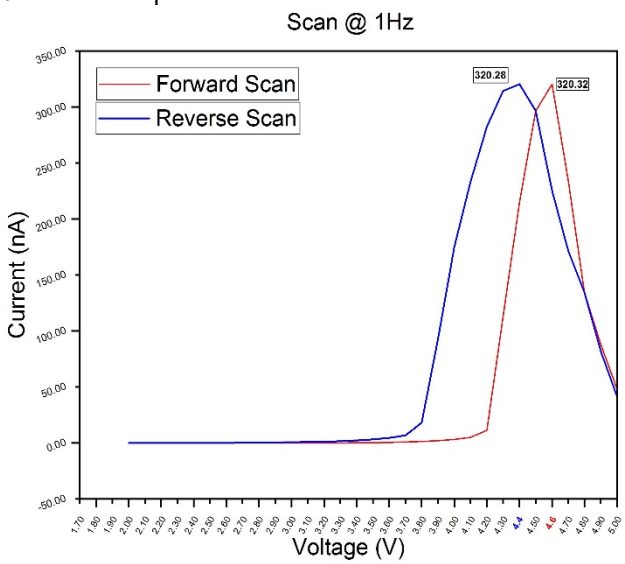


Figure 4: Scan of the voltages at 1 Hz.

In the same way, the creep phenomenon is not considered to be disturbing the regulation of the algorithm because it is known that is a slow drift response that appears some milliseconds after the voltage is applied. Furthermore, it has not the same creep effect two signals with different history [4]; i.e., a signal from 0 to 5 V than a signal from 3 to 7 to 5 V.

FIRSTS TESTS ON THE BEAMLINE

The tests have been carried out in the BL16 NOTOS beamline to show the performance of the ALBA Em# as monochromator controller. It is worth to note that the current algorithm is only focused on maximizing the current of the ion chamber, acting as an equivalent to what is called oscillation mode in the monochromator controllers that are already being used at ALBA. The purpose is to deal with two of the common monochromator issues mentioned previously: the mechanics inertia and the parallelism of the crystals. The third one, the temperature drift, is out of the scope of the regulation since, due to its nature, this behaviour is much longer in time (around hours).

The test was executed by performing a qExafs (quick Extended X-ray Absorption Fine Structure) scan where it can be visible the regulation of the algorithm by superimposing a reference scan (without the Em# new functionality) and the regulation one. Prior to launching the energy scan, a rocking curve is used to align the pitch crystal within the stepper motor range, on top of which the piezo is actuating.

In Fig. 5 it is shown the results of the scan of an energy range from 5000 eV to 12000 eV, an acquisition of 3500 points and integration time of 0.1s. The scale of the current is in the range of hundreds of nA since a filter of Aluminium 10 µm thick was inserted. In these scans, to move within the energy range it was not used the perp

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP6

motor, the one that changes the crystal gap, to easily compare the performance of the Em# regulation.

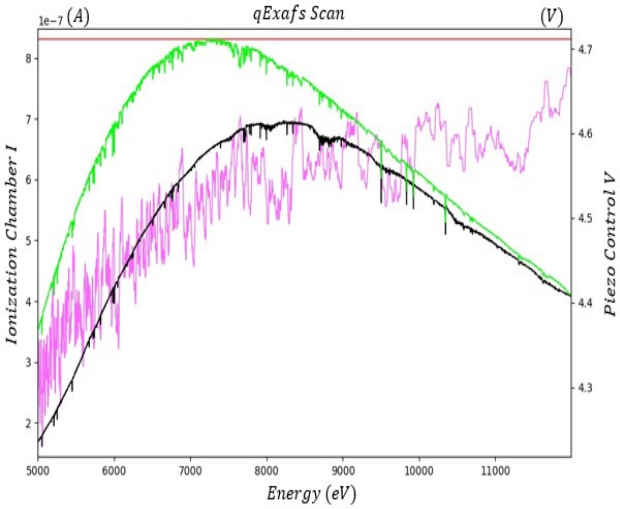


Figure 5: Energy scan with and without algorithm.

Figure 5 shows that the Em# regulation, green curve, is maximizing the current in the first half of the scan while maintaining it in the second half with respect to the reference scan (black curve for current and red curve for the steady voltage). As the energy is increasing, the current also increases and the controller maintains this tendency by adjusting the crystal inclination (pink curve) to compensate any disturbance inherent to the motion (bragg motor to increase the energy) and any instabilities of the mechanics. However, this regulation is clearly introducing noise to the system as well. Since the measurements of the electrometer are punctual, any noise added to the measurement can fake some value so the algorithm will make the piezo to oscillate. In order to deal with this disturbance, a fifo queue was implemented in the algorithm to average an amount of values (the user can set this window size). Besides the noise in the signal, the downwards peaks of diffraction (inherent to the crystal of the monochromator) seen in the black curve also disturbs the regulation. This intrinsic artifact is not aimed to be corrected, in fact, a filter to ignore compensation of these glitches is implemented. By getting the standard deviation of the fifo and comparing if the difference, maximum minus minimum values in the queue, is greater than the standard deviation times a factor (that needs to be set), the algorithm will ignore this value and, hence, not regulate.

CONCLUSION

After the initial tests on the beamline of this new feature of the Em#, it has been proved that the device can act as a versatile element.

The performance of the monochromator controller belongs to a preliminary design that establishes the proof of concept and gives an insight of its potential use for beamline operation in the future. This implementation will solve the problem of stock of this type of controllers and will ensure a continuous support taking advantage of the in-house foundation of the Em#. Furthermore, for the same reason, it will ease the integration of the controller within the ALBA control system.

It was observed that it is achieving to maximize the current over the energy range, but at the expense of introducing noise. Another drawback of the actual implementation is that the parameters of the algorithm needs very accurate fine tuning and can vary within the environment (i.e. beamline).

The functionality is designed with the possibility of having a variable delta step, but it was not implemented due to the initial stage of the project. It is still in development stage, where the main efforts will be focused on reducing noise. As a first approach, a hardware filter is aimed to be implemented in the input of the piezo driver to lessen the noise introduced by applying the voltage from the electrometer. Modification of the algorithm will be done regarding the same objective as well as changing the measurement of the electrometer by oversampling the acquisitions. More tests will be performed on different energy ranges and conditions, as the use of the perp while moving in energy. Further steps will be done to explore other types of regulation and on the integration of the new functionality in the control system.

ACKNOWLEDGEMENTS

Thanks to the people involved in the development of the whole Em# project. Thanks to the computing division members that contributed to the definition, execution and testing of the present monochromator controller project. Thanks to the NOTOS beamline scientists, Giovanni Agostini and Carlos Escudero, for providing the beam time and support needed for the tests of the monochromator controller within the beamline operation.

REFERENCES

[1] X. Serra-Gallifa *et al.,* "Guaranteeing the Measurement Accuracy in Em#", in *Proc. PCaPAC'18*, Hsinchu City, Taiwan, Oct. 2018, pp. 216-219. doi:10.18429/JACoW-PCaPAC2018-THP22

[2] Murari Lal Azad *et al.*, "P&O algorithm based MPPT technique for solar PV System under different weather conditions", in *Proc ICCPCT'17,* Kollam, India, 2017. doi: 10.1109/ICCPCT.2017.8074225

[3] W. Zhu and X-T. Rui "Hysteresis modeling and displacement control of piezoelectric actuators with the frequency-dependent behavior using a generalized Bouc–Wen model", *Precis Eng*, vol. 43, pp. 299-307, 2015. doi:10.1016/j.precisioneng.2015.08.010

[4] Hewon Jung and Dae-Gab Gweon. "Creep characteristics of piezoelectric actuators", *Rev. Sci. Instrum.*, vol. 71, pp. 1896-1900, 2000 doi:10.1063/1.1150559

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP7

OPC UA BASED USER DATA INTERFACE AT ELBE

K. Zenker*, M. Justus, R. Steinbrück
Institute of Radiation Physics, Helmholtz-Zentrum Dresden-Rossendorf, Dresden, Germany

Abstract

This paper presents an OPC UA based data interface implemented at the high-power radiation source ELBE. It provides access to ELBE machine data via dedicated gateway devices.

In addition to the intrinsic security and authentication features included in OPC UA an access control mechanism was implemented at the PLC level. This allows to enable/disable user data access using the SCADA system of ELBE.

INTRODUCTION

The center for high-power radiation sources ELBE delivers different kinds of radiations serving a variety of user groups. The facility is in operation for more than 20 years and over time different machine interfaces were created for different user groups tailored to their needs. On the one hand some user groups provide experiment data like counting rates to the operators. This information is used for machine optimization. On the other hand some user groups have the permission to change certain machine parameters, like undulator gap size or beam repetition rate on their own.

So far different solutions have been used to provide those machine interfaces. Solutions include direct access to the SCADA system via dedicated accounts and OPC interfaces available in the SCADA system.

Here we present an approach of providing a common machine and ELBE data interface based on the OPC UA standard [1]. It is applicable to all use cases at ELBE and can be accessed via a single access point OPC UA server.

In the following, the infrastructure at ELBE is introduced. After the ELBE data interface (EDI) and its architecture is discussed.

ELBE INFRASTRUCTURE

ELBE is operated using the SCADA system WinCC by Siemens. The majority of ELBE systems is connected to WinCC via industrial Ethernet and proprietary S7 communication. At control level SIMATIC S7 300 and S7 400 Programmable Logic Controllers (PLC) are used. Different subsystems, like the machine protection system or the personnel safety system, and different subsets of each subsystem, belonging to e.g. different beam line sections, are implemented on dedicated individual PLCs.

The integration of subsystems based on MicroTCA.4 [2] hardware, which do not provide S7 communication interfaces, into the existing infrastructure is based on OPC UA using the open source C++ software toolkit ChimeraTK [3, 4]. Here, OPC UA allows a direct integration into WinCC using the native OPC UA support of WinCC.

* k.zenker@hzdr.de

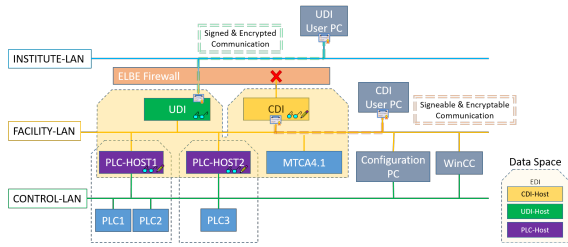


Figure 1: Overview of the ELBE data interface (EDI). The only available access points are the gateways named UDI and CDI. All other components are hidden to EDI users.

Commercially available OPC UA gateways [5] (UA Link) by IBHsoftec are used at ELBE as a data bridge between the Siemens S7-300/400 PLCs and OPC UA based applications as described in [6–8]. These UA Links are the basis of EDI.

ELBE DATA INTERFACE ARCHITECTURE

As introduced above two data sources – direct OPC UA sources like MicroTCA based subsystems and indirect OPC UA sources that aggregate PLC data via UA Links – need to be considered for EDI. Both can be interfaced by UA Links, which allows to implement EDI based on the UA Links.

From a traffic point of view it is desirable to place the UA Links as near as possible next to the PLCs in respect to the control network. Where possible the gateways have been directly connected to the according PLC interfaces. This automatically led to the introduction of several so called PLC-Host gateways throughout the ELBE machine and a multi-layer structure as introduced in the following.

Abstraction Layers

Figure 1 shows an overview of the EDI architecture currently implemented at ELBE. In a first abstraction layer of EDI are the PLC-Hosts, that are shown in the bottom part of Fig. 1. Depending on the location and the address space of the PLCs data of multiple PLCs can be aggregated by a single UA Link (see Fig. 1 PLC1 and PLC2) or a single UA Link per PLC-Host is used (PLC3 in Fig. 1). At this level the OPC UA address space is automatically generated based on the symbols defined in the PLC STEP7 project.

In general, the configuration of a UA Link allows to define the access level for each process variable on the OPC UA side. Additionally user accounts can be configured to have read only or read write access. However the available process variables are the same for each user and single process variable access cannot be defined user dependent. Furthermore, it is not possible to change process variable access during runtime.

Since UA Links at this level are not visible to the outside world it is possible to waive encryption and authentication

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP7

UDI	CDI	data structure	type	description
X	X	CURRENT.actual_value	Float	actual read back current
X	X	CURRENT.max_value	Float	maximum possible current value
X	X	CURRENT.min_value	Float	minimal possible current value
X		CURRENT.setpoint	Float	current setpoint
	X	CURRENT.SETPOINT_RW.accessible	Boolean	TRUE == write access granted
	X	CURRENT.SETPOINT_RW.value	Float	current setpoint
X	X	CURRENT.unit	String	engineering unit
X	X	STATUS.on	Boolean	TRUE == magnet current controller switched on
X	X	STATUS.ready	Boolean	TRUE == magnet current controller setpoint reached

Figure 2: Example address space of a magnet. The left two columns indicate which variables are available via the control and user data interface respectively.

for the sake of performance and ease of maintenance. They are configured to allow read and write access to PLC data.

In a second EDI layer, data from the first layer is aggregated and remapped. Here only two UA Links are present – one acting as user data interface (UDI) and the other acting as control data interface (CDI). The general difference between UDI and CDI is that UDI provides only read access to the UA Links in the lower layer, whereas CDI provides read and write access. In detail this results in a user friendly well structured address space removing any footprints from the PLC project symbols. An example can be seen in Fig. 2, which shows the data structure of a single magnet. Here it becomes clear that the address space for read only process variables is shared between UDI and CDI, whereas dedicated process variables for the current setpoint exist for UDI and CDI. This results from the PLC implementation introduced below.

The address space of UDI and CDI is created in a structured manner using UA Link local variables exclusively. They are bound to according OPC UA variables published by the PLC-Host level.

User Data Input

Additional local variables are defined at the UDI level, which are not bound to PLC-Host level variables. They are writable by ELBE users to provide information of interest for the ELBE machine operation. E.g. counting rates measured by the users are published like that and used to optimize the machine state with respect to a maximum counting rate.

User Authentication

The access point of UDI only provides endpoints that use encryption and user authentication. At ELBE a Public-Key-Infrastructure (PKI) including a ROOT Certification Authority (CA) and dedicated user group CAs has been set up that hand out user certificates based on the X.509 standard. This reduces the management effort related to the UA Links, where only the CA certificates and corresponding revocation lists need to be installed.

As can be seen in Fig. 1 CDI is currently only available for ELBE internal use inside the closed machine network. Here an endpoint without encryption and user authentication is provided. This is beneficial for debugging purposes. Once users will use CDI the endpoint will be removed and the user authentication will be similar to the one used with UDI.

PLC LEVEL IMPLEMENTATION

The PLC implementation had to respect the fact, that the ELBE machine code has been permanently extended by different developers over the last 20 years. This results in a heterogeneous data structure in respect to symbols types and scaling. To transform this source of data into a homogeneous representation it has been necessary to map the data into a well structured format already on PLC level. Even when this requires additional resources it is reasonable for the following reasons:

- Scaling information is best available and documented within the PLC project
- Data collection by the UA Links is completely transparent to the PLC developer
- The concentration to a known EDI data interface allows for easy debugging and disabling in case of machine problems
- A known symbolic data structure at PLC level allows to automate major parts of the UA Link configuration
- Misconfigurations only affect data introduced by the EDI development and not parts of the well tested present PLC code

In detail, three components are implemented for EDI at PLC level, that are introduced in the following.

Data Type and Unit Conversion

The first component converts register types and PLC internal units to physical units. The type conversion is needed because e.g. different magnet interfaces provide readbacks as different data types given in different units. Using the conversion component a consistent machine data representation can be created and exposed to users.

Write Access Logic

The second component implements write access of EDI. This access type adds significantly more complexity to EDI compared to read-only access. First of all in contrast to the read-only access, where the PLC register is simply duplicated, here two additional registers (B and C) per register to the published (A) are added. This is shown in Fig. 3. One (register B) is used to transfer data from the PLC to EDI and the other one (register C) is used to transfer data from EDI to the PLC. The PLC permanently checks for data changes of the register C and register A. Depending on whether a change in A or C is registered two different data paths are possible.

The first path is activated once a data change of register A is detected. In that case the data of register A is copied to register B, which is mapped to a corresponding process variable B' in one of the PLC-Hosts. The latter connection is a bidirectional one, which is fixed by the UA Link implementation. In the following the data is transferred to CDI via an unidirectional connection. This is possible by defining a local variable on the UA Links used for CDI. For local

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP7

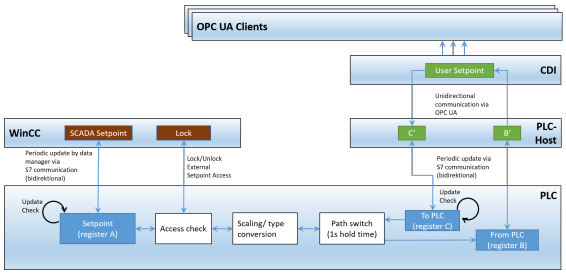


Figure 3: Write logic schematic used at ELBE. Details are given in the text.

variables it is possible to define data sources - in this case the data source is the process variable A'. At this point the data change on the PLC is visible to the user of CDI. At the same time data sinks that are to be filled on data changes of the local variables can be defined. This is used to define an unidirectional connection to the process variable C' of the PLC-Host that is connected bidirectionally with register C. The connection of C to A is interrupted as long as updates are transferred to B. It can only be closed 1 s after the last update of register A. This time is in principle defined by the time needed to transfer data from register A to register C, which includes network latency.

The second path is activated when the local variable accessible to the users is changed and the first path is not active. Now the data is transferred via register C' to C and finally copied to register A. Once this path is active no data is transferred from register A to B. This connection is interrupted as long as updates are received via register C and can only be closed 1 s after the last update of register C.

Using the approach of having two data paths and a lock time of 1 s for each direction was found to be a valid configuration at ELBE used to avoid data reflections. In case of the first path a reflection would result in overriding register A with preceding values that are transferred from register B to C. During this transfer the registers A and B could have been already updated. Similarly, in case of the second path data transferred from the CDI UA Link via register C', C, A, B and B' to CDI could override the data entered by the user in the meantime with old data.

However, our approach does not prevent data clashes resulting from different users using CDI at the same time or that result from simultaneous data changes introduced by the SCADA system, which always writes directly to register A.

Access Control

The third component is used to make sure that the SCADA system always has full control of ELBE. It adds information and controls to the SCADA system that allow to enable/disable EDI at all or just parts of it. In order to do so different groups are defined. Those groups represent different beam line section of the ELBE machine and correspond directly to subsets of the address space created in the UDI and CDI. In the end the component copies data from the registers created by the write access logic component only if the corresponding access bit is set via the SCADA system. The current access status is published to CDI via a corresponding process variable, e.g. `CURRENT.SETPOINT_RW.accessible` in Fig. 2. In addition, indicators visualizing active data changes by CDI are added to the SCADA system. This allows to distinguish between data changes by CDI and SCADA clients used outside of the control room.

PLC Timing

The components introduced above are added to a user code block of the PLC that has a fixed maximum cycle time of 150 ms. We observed an increase of cycle time by 20 ms caused by introducing EDI. This time is dominated by the write access logic related to CDI. However, the total current cycle time is 40 ms, which shows that the cycle time is still well below the limit. The Timing of time critical code executed by interrupts and high priority tasks is not affected by CDI.

CONCLUSION

At the center for high-power radiation sources ELBE an OPC UA based data interface has been established. It allows direct external access to ELBE data, which was not possible before. E.g. now users can integrate ELBE data into the experiment data acquisition or record ELBE data directly along with their experimental data. Another application of EDI is the implementation of slow feedbacks. This was successfully demonstrated in case of a beam position stabilization used to fixed the beam at sub-millimeter level on a target for a period of a week in a high power beam experiment [9, 10].

The interface is based on commercially available gateway devices that are used in a multi-layer approach. This allows to create a single access point with a configurable address space independent of the address space used at PLC level or by interfaced OPC UA servers of integrated subsystems.

On the one hand the connections to the data interface are secured by using user authentication and encrypted connections. Encryption is based on user certificates and dedicated OPC UA endpoints. A PKI for handling of user certificates was created. On the other hand the data interface is fully integrated to the ELBE SCADA system such that ELBE operators always have the full control of ELBE. This is ensured by indicators of data accesses via the data interface and the option to block selectively parts of the write accesses by data interface or to disable the write access of the data interface at all.

The PLC level implementation increased the corresponding code block cycle time by 20 ms, which left the total cycle time well below the maximum defined cycle time.

REFERENCES

[1] *OPC Unified Architecture*, The OPC Foundation. `http://opcfoundation.org/opc-ua/`

[2] PICMG, *PICMG specification MTCA.4 Rev. 1.0*, Aug. 2011. `https://www.picmg.org/openstandards/microtca`

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP7

[3] M. Killenberg *et al.*, "Abstracted Hardware and Middleware Access in Control Applications", in *Proc. ICALEPCS'17*, Barcelona, Spain, Jan. 2018, pp. 840–845. doi:10.18429/JACoW-ICALEPCS2017-TUPHA178

[4] G. Varghese *et al.*, "ChimeraTK - A Software Tool Kit for Control Applications", in *Proc. IPAC'17*, Copenhagen, Denmark, May 2017, pp. 1798–1801. doi:10.18429/JACoW-IPAC2017-TUPIK049

[5] *IBH Link UA*, IBHsoftec. https://www.ibhsoftec.com

[6] R. Steinbrück *et al.*, "Control System Integration of a MicroTCA.4 Based Digital LLRF Using the ChimeraTK OPC UA Adapter", in *Proc. ICALEPCS'17*, Barcelona, Spain, Jan. 2018, pp. 1811–1814. doi:10.18429/JACoW-ICALEPCS2017-TUPHA178

[7] K. Zenker *et al.*, "MicroTCA.4-Based Low-Level RF for Continuous Wave Mode Operation at the ELBE Accelerator", *IEEE Trans. Nucl. Sci.*, vol. 68, no. 9, pp. 2326–2333, 2021. doi:10.1109/TNS.2021.3096757

[8] K. Zenker, M. Kuntzsch, and R. Steinbrück, "Integration of OPC UA at ELBE", in *Proc. ICALEPCS'21*, Shanghai, China, Mar. 2022, paper TUPV010, pp. 400–404. doi:10.18429/JACoW-ICALEPCS2021-TUPV010

[9] HZDR press release, *Producing medical isotopes at extreme energy density*, Feb. 2022. https://www.hzdr.de/db/Cms?pOid=65365&pNid=3438

[10] DEMCON press release, *Medical radioisotopes produced with the world's most power-dense reactor*, Feb. 2022. https://dam.demcon.com/medical-radioisotopes-produced-with-the-worlds-most-power-dense-reactor

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP8

IC@MS — WEB-BASED ALARM MANAGEMENT SYSTEM

Ł. Żytniak*, M. Gandor, P. Goryl, J. Kowalczyk, M. Nabywaniec, S2Innovation, Cracow, Poland
S. Rubio-Manrique, ALBA Synchrotron Light Source, Cerdanyola del Vallès, Spain

Abstract

The IT world is moving to the web and cloud. IC@MS is a web-based alarm management system. Every control system can face unexpected issues, which demand fast and precise reactions. As the control system starts to grow, it requires the involvement of more engineers to access the alarm list and focus on the most important ones. IC@MS allows the users to access the alarms fast, remotely via a web browser. According to current trends in IT, creating a web application turned out to be the most comfortable solution. IC@MS is the extension and web equivalent to the Panic GUI desktop application. There is no need to install it on the client's computer. The access to the different functionalities can be restricted to the users provided just with appropriate roles. The web-based alarm management system provides a better user-friendly user interface for everyday use with Integration with Active Directory. Alarms can be easily added, edited, and managed from the web browser*. It has a Web API that can be used by 3rd party applications. The instance of IC@MS is available on Amazon Web Services (AWS) and Microsoft Azure clouds.

ALARM SYSTEM IN TANGO

Tango Controls is a free open source device-oriented controls toolkit for controlling any kind of hardware or software and building SCADA (Supervisory Control And Data Acquisition) systems (Fig. 1). Tango Controls is operating system independent and supports C++, Java and Python for all the components. As Tango Controls exists for more than 20 years and becomes more and more popular among facilities, it has proved itself as a reliable toolkit [1]. Engineers at institutes like particle accelerators are daily dealing with hundreds or thousands of signals per second coming from different types of devices. Therefore, there is a need of an application to detect and monitor non typical situations. One of the most popular solution is PANIC and PyAlarm Device Server.

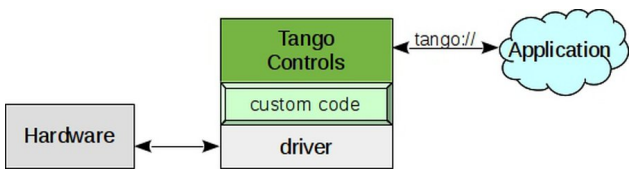


Figure 1: Tango high level overview.

PANIC

PANIC (Package for Alarms and Notification of Incidences from Controls) is an alarm system developed in ALBA Synchrotron. It is a set of of tools that provides:

- Periodic evaluation of a set of conditions.
- Notification (email, sms, pop-up, speakers)
- Keep a log of what happened. (files, Tango Snapshots)
- Taking automated actions (Tango commands / attributes)
- Tools for configuration/visualization

The Panic package contains the python AlarmAPI for managing the PyAlarm device servers from a client application or a python shell. The panic module is used by PyAlarm, Panic Toolbar and Panic GUI [2].

PyAlarm

The key element of PANIC toolkit is PyAlarm device server. This device server is used as a alarm logger, it connects to the list of attributes provided and verifies its values [3]. That features are needed to ensure comfortable alarm management for user. Each alarm is independent in terms of formula and receivers, all alarms within the same PyAlarm device will share a common evaluation environment determined by PyAlarm properties.

Panic GUI

Panic GUI is a application written using Taurus library. It allows to check existing alarms and manipulate them. Panic GUI is an desktop application for controlling and managing alarms. It depends on panic and taurus libraries. It allows the user to visualize existing alarms and adding/editing/deleting alarms. In edit mode user can change name, move alarms to another device, change descriptions and modify formulas. Additional widgets in which the app is equipped allows alarm history viewing, phonebook editing and device settings manipulation.

EPICS

EPICS (Experimental Physics and Industrial Control System) is a set of Open Source software tools, libraries and applications developed collaboratively and used worldwide to create distributed soft real-time control systems for scientific instruments such as a particle accelerators, telescopes and other large scientific experiments. Moreover, it is a set of software tools and applications which provide a software infrastructure for use in building distributed control systems (Fig. 2). Such distributed control systems typically comprise tens or even hundreds of computers, networked together to allow communication between them and to provide control and feedback of the various parts of the device from a central control room, or even remotely over the internet [4].

* lukasz.zytniak@s2innovation.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP8

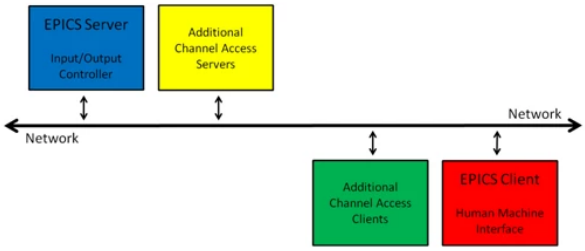


Figure 2: EPICS Network Block Diagram.

IC@MS

Advantages of using IC@MS (Integrated Cloud Ready Alarm Management System):

1. Better user interface for everyday use,
2. Access from anywhere through web browser,
3. No need to install on the client's computer,
4. REST API,
5. Multiple access roles,
6. Integration with Active Directory (AD),
7. SMS, mobile or mail notifications,
8. Fast and effective – tested with thousands of alarms.

The IC@MS provides the same functionalities as Panic GUI but also extends it with some new functionalities like adding devices supporting different protocols (Fig. 3). What is important, IC@MS can be integrated with both TANGO and EPICS control system.

KEY FEATURES

Main IC@MS features:

- Alarms list,
- Management of existing alarms,
- Multiple sources of data: EPICS PV, HTTP, MQTT,
- New alarm definition,
- Alarms history browser,
- Multiple searching and filtering options,
- Documented REST API (Swagger).

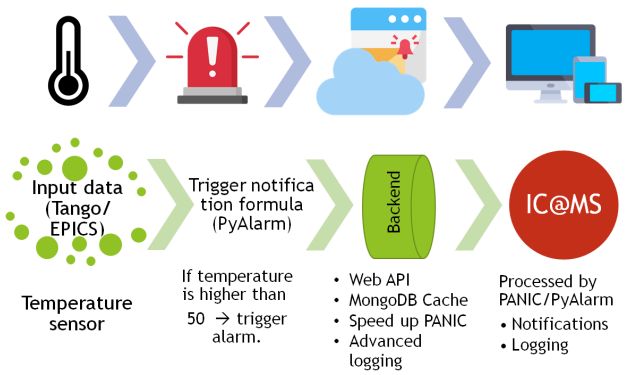


Figure 3: IC@MS Overview.

Cloud Ready

The IC@MS can be simply deployed to the cloud like Amazon AWS or Microsoft Azure. It lowers operating costs. Moreover, there is no risk connected with server security and maintenance. The application is accessible via a web browser with only an internet connection.

Alarm Dashboard

The alarm dashboard (Fig. 4) is the view that users see most often. That dashboard presents active and not active alarms. When the alarm is triggered, its color changes following the severity. The most important and recent alarms are shown on top. There are buttons to perform operations like acknowledge, reset or disable the alarms.

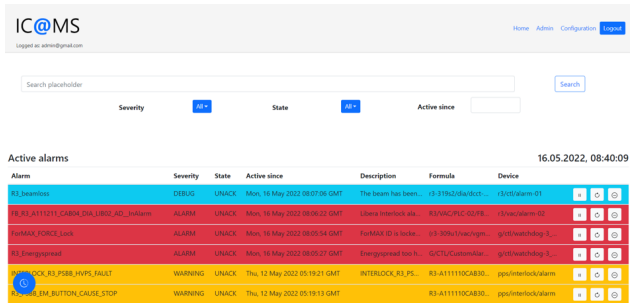


Figure 4: Alarm dashboard in IC@MS.

User can modify existing alarms (Fig. 5) and create new one using the configuration page. To create new alarm user should provide:

1. Alarm name,
2. Receivers (email / phone number),
3. Formula (defines when the alarm will be triggered),
4. Severity.

Fields can be edited.

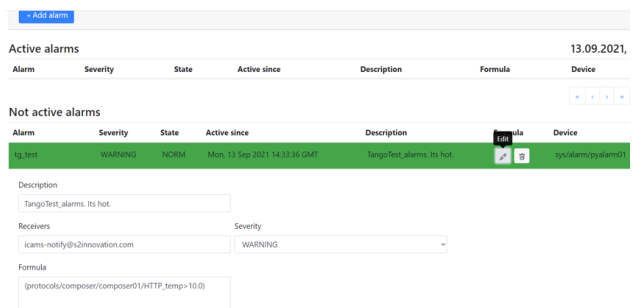


Figure 5: Creating alarm in IC@MS.

Users And Roles Management

Security is priority, therefore, only registered users can login and manage the alarms. Moreover, to make work with alarms easier, the IC@MS has roles, the user can see easier alarms that he is interested in. Additionally, with Active Directory (AD) users can use the same credentials.

Data Sources and Composers

The configuration page, has option to define new data sources and composers devices (Fig. 6).

Data sources:

- HTTP,
- MQTT,

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP8

- Modbus.

Composers:

- Propagate signal from low level device.
- Compatible with HTTP, MQTT, Modbus.
- Extract data from different data formats: JSON, XML, HTML, raw strings, Modbus.

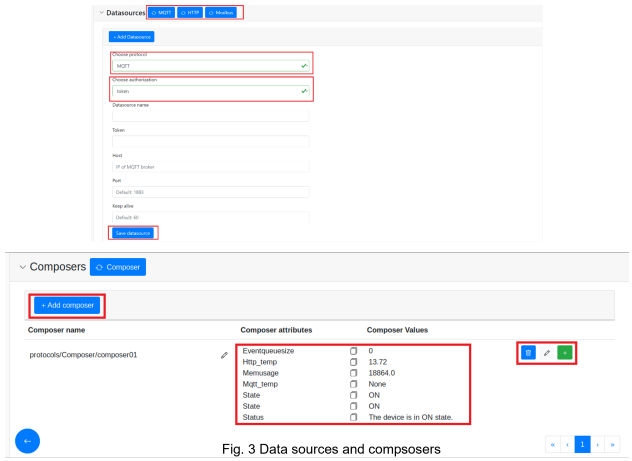


Figure 6: New data sources and composers.

ALARM HISTORY

One of the most useful functionality is alarm history, which allows users to view the history of alarms including dates, states, and values of formulas connected to the selected alarm (Fig. 7). Moreover, users can browse alarms by date, severity, and other criteria (Fig. 8).

CONCLUSION AND FUTURE WORK

IC@MS (Integrated, Cloud-ready @larm Management System) fulfils all requirements of modern and robust alarm systems. During the tests at particle accelerator facilities, it proves that can process thousands of alarms. IC@MS is continuously improved with feedback from the users.

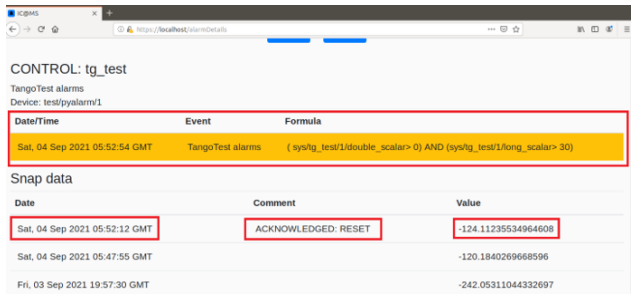


Figure 7: Single Alarm history.

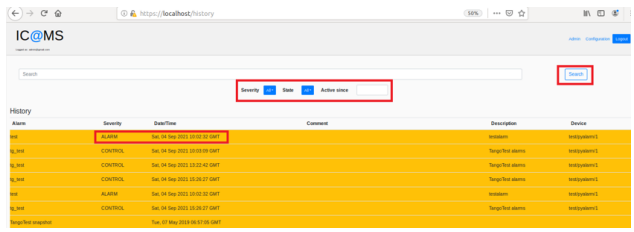


Figure 8: All alarms history.

REFERENCES

[1] TANGO Controls System website, https://www.tango-controls.org/

[2] PANIC documentation, https://tango-controls.readthedocs.io/projects/panic/en/latest/

[3] PyAlarm documentation, https://www.tango-controls.org/developers/dsc/ds/1401/

[4] EPICS Documentation, https://docs.epics-controls.org/en/latest/

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP9

SIMPLE PYTHON INTERFACE TO FACILITY-SPECIFIC INFRASTRUCTURE

J. Gethmann*, E. Blomley, P. Schreiber, M. Schuh, W. Mexner, A.-S. Müller
Karlsruhe Institute of Technology, Karlsruhe, Germany
S. Marsching, aquenos GmbH, Baden-Baden, Germany

Abstract

The various particle accelerators hosted at KIT represent a complex infrastructure with a live control system interface, a data archive, measurement routines, and storage and management of metadata, among other aspects. The "KIT Accelerator Python tools" were created to provide a unified interface to all aspects of the accelerator infrastructure for both short-term student projects and basic accelerator operations. Instead of creating another custom framework, these sets of tools focus on bridging the gap between well-established libraries, our facility and accelerator specific needs. External and accelerator specific libraries are glued together to provide an interface in order to minimise the technical knowledge of the accelerator infrastructure needed by the end user. Best practice software engineering workflows of continuous integrationn were implemented to provide automatic testing, packaging, API documentation and release management. This paper discusses the general motivation and approach taken to create and maintain such a set of Python modules.

INTRODUCTION

At KIT, we operate multiple kinds of accelerators (KARA, FLUTE) and further compact accelerators are in the planning and building phase. KARA, a 2.5 GeV storage ring, a part of the KIT Light Source, with a circumference of 110 m. FLUTE [1] includes a linear accelerator. Further accelerators are planned, e. g. a plasma accelerator and a compact storage ring within the project cSTART [2]. This paper first describes the setup of the controls infrastructure, then introduce the requirements for accelerator physics' experiments and the control of the machines. Then, the strategies to solve issues and the decisions made are described. Lastly, we present some lessons that we learned in the process.

SETUP

The controls infrastructure has to integrate the required different operation modes of the different KIT accelerators with a quick and safe access by e. g. non-experienced students via users of synchrotron radiation to experts with very demanding accelerator physics and technology experiments.

The data flow starts by providing live data and control for machine parameters from magnet power supplies to diagnostics instrumentation. Relevant parameters are archived in databases that are flat for performance reasons and are furthermore highly available. Snapshots of machine settings can be saved to and restored from relational databases for selections of groups, e. g. the pre-accelerator synchrotron, in short booster, settings can be restored independently from the storage ring's settings. Furthermore, there exist electronic lab notebooks for routine operation and certain typical experiments. For machine optimisation purposes and typical measurements needed for beam dynamics simulations, MATLAB® routines write their outputs to a central file storage.

Though the controls architecture for the different accelerators is similar, there are logical stand-alone systems for each accelerator requiring different addresses and namespaces.

REQUIREMENTS

For different stakeholders, the requirements differ. Especially for students and visiting researchers, it is necessary to get easy and quick access to certain sub-systems without in-depth knowledge of how exactly these systems are set up or interact with each other. They do not need to know from which control system component the data originates nor which protocol that systems speaks. Another crucial point is a quick setup, possibly with limited permissions granted. Facility scientists need access to a broad variety of data and therefore have to interact with many systems at the same time. However, the knowledge of the specifics of each particular system is typically not required. Finally, there are the developers of such systems who want to have maintainable and flexible code, which is reusable and actually used. This includes to adopt FAIR principles. We need straightforward access to all systems without the domain knowledge of the individual sub-systems or the infrastructure's topology.

In the past, we struggled with these different user groups trying to solve similar issues with their own individual scripts. Such scattered scripts were often written in different languages, with different levels of code quality and focusing on different aspects of the same task. This resulted in making these scripts very hard to share, maintain and re-use, especially across different user groups. The user experience of systems with different interfaces for similar behaviour is not good and can lead to errors in the worst case. Lastly, substantial inside knowledge is required to be aware of such scattered scripts and where they can be found.

APPROACH

Instead of creating another framework or a generic scientific library, our approach bundles existing ones for convenient ways to use them for the KIT accelerators according to our needs.

* gethmann@kit.edu

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP9

In general, our packages provide an abstraction layer towards the various libraries. We initialise them with site-specific settings in advance, like server addresses and configuration settings, because the user does not need to know this information. In addition, changing of backend libraries can be done transparently for the user with this approach. Furthermore, the user does not need to know if some parts of the infrastructure moved to a different location, e. g. because of a new URL for a new version of a REST-API.

Nowadays, distributable well-structured Python packages can be created comparably easily with bootstrapping tools like PyScaffold. This enables one to use packages instead of a bunch of more or less sorted Python scripts where you cannot handle dependencies on each other or on third party packages well, unless you have only a single repository, which includes everything. The latter is not what we want, especially as some packages depend on large libraries like *scikit-learn* or software that is either hard to set up and configure or serves only a few special needs. Installing such software should neither be mandatory for users nor required on embedded systems. With our approach, we can use these packages for different use-cases such as pre-processing for machine learning pipelines on the KIT computer clusters, for the daily needs on one's office workstation, as well as for developments inside the accelerator control system. Especially the latter will be much more important in the near future with the increasing usage of Python SoftIOCs [3].

For office and data science applications, distributable Python packages are the preferred format for installation as the installation procedure of such is operating system-independent. It is already known to many users and usable without additional permissions beyond network access. Nevertheless, for the internal accelerator networks we prefer to build our own Debian packages.

Style guides can help to get on board as a person unfamiliar with the respective code base. However, they are arbitrary in some way and may result in so-called "bike shedding" discussions, so we took *black*'s approach [4] and used a set of well-established linters, formatters and style guide checkers. Providing feedback on the formatting, linting and API documentation already at an early stage helps us to maintain readable and documented code. Good and common tooling provides automatic formatting. Code review of consistently formatted code with documentation is far more enjoyable and lets one focus on the logic rather than the style.

We try to have a high test coverage to ensure reliable productive code and to avoid re-introducing bugs, which have already been fixed in the past. Furthermore, tests can provide examples of the intended use, if no example code exist. We use *pytest* [5] with the *coverage* plug-in for testing as it makes testing more convenient than the build-in *unittests* and might nudge students, scientists and other users into writing tests for their personal code.

We host the packages and documentation on a website as they are the interface for most of the end users, who use the libraries but do not develop them. We plan to integrate this functionality into our GitLab setup.

DECISIONS

Python Packages

Nowadays, Python is the eco-system of choice both for (data-)scientists and for many software developers. The scientific libraries eco-system is extensive, thanks to its proper C-API. As it is open source and provides multiple programming paradigms, including object orientation, it is very popular among software developers. Hence Python is a good language to wrap existing APIs to control and archiving systems. We developed two kinds of packages: data aggregation wrappers and helper libraries. The first one is for

- archived data,
- electronic lab notebooks,
- live data,
- our settings database, and
- for general data formats and access of different kinds.

They wrap the different data sources and provide coherent and simple interfaces with sane default settings for our facility.

The second one, the helper libraries take care of the interaction between the other libraries and selection of the correct settings for the accelerator of choice.

Besides the Python packages, we provide Debian packages for the in-use Ubuntu LTS systems. These systems are part of our accelerator network and thus are well defined, have no connection to the internet, and no packages are meant to be installed by local users, consequently *pip* is not installed. That is why, we build our Python packages with all their dependencies that are not part of existing Ubuntu repositories.

Development Workflow

Changes to the tooling are made in a central management repository, and an automated workflow takes care of synchronising these changes across all packages. Each project can be separated into a tooling part, which is very similar for each project, and the actual code part. For creating a new package, we have got an extension for *PyScaffold*, which takes care of setting up the tooling part. In the case the tooling is adjusted, a central management repository automatically takes care of keeping it synchronised across the packages.

For existing projects, people can contribute in various ways, depending on their experience and the effort they want to invest; filing bugs or requesting features in the issue tracker, fixing small bugs via a web-UI or working on a locally cloned project. In the latter case, it is advised to install *pre-commit* [6]. This tool activates a set of Git pre-commit hooks that enforce our style guide. The pre-commit hooks run on the client side and already fix the code, or at least warn the developer about issues, which would lead to a failure in the continuous integration test pipeline.

On the server side we use GitLab, which is hosted on premise with access to our internal data sources, so we can run tests that can rely on these internal APIs. GitLab provides an issue tracker, continuous integration (CI) infras-

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THPP9

tructure, a Docker registry, a Python repository, a proper web-editor for small changes and previewing Markdown or ReStructuredText, and might be used for serving documentation and Debian packages in the future. This ensures that both, users and maintainers, only have to deal with a single platform.

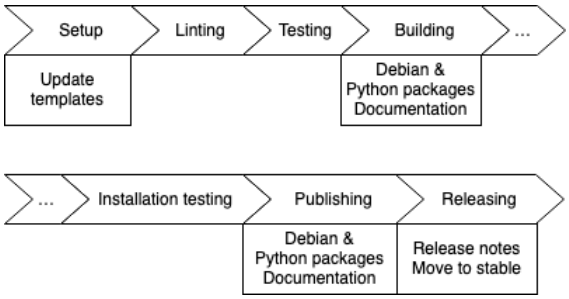


Figure 1: GitLab continuous integration. Besides the usual linting, testing, building, releasing, there is a setup, installation testing and publishing step. Besides some steps act on Python and Debian packages as well as documentation.

For the continuous integration we make use of custom Docker containers with pre-installed environments for the individual steps of the pipeline. The containers are managed in a dedicated project alongside the actual packages. Figure 1 shows the workflow. In the setup step we take care of updating the configuration and the template files of the project. As it is done for every new merge request, the different projects do not diverge in their overall structure and style. In the linting and testing part, checks are done again to enforce that code on the main branch is as easy to understand as tooling can guarantee. The build step not only builds the Python packages, but also the Debian packages and artifacts such as the documentation in HTML. In that way the code reviewer can study such tangible by-products and artifacts. As we build Debian packages, which might have dependencies on further packages, we have a dedicated "installation testing" step as part of the pipeline. During that stage the Debian packages are installed and tested as done during an update and as done in a green field, to ensure that conflicting dependencies do not disrupt or stop the processes on the production computers in either of these scenarios. The last step of the merge pipeline is to publish the packages and documentation to our internal repositories.
We try to create merge requests and code reviews for all of our code, so that more people are familiar with the code base. Best practices are taught and releases of malfunctioning code are minimised.

The release step is a dedicated manual step including a major, minor or patch release decision. Compiling the release notes, tagging the commits and moving the repositories to the stable branch is then done automatically. As an additional security measure, our control-system computers use an Apt repository that does not automatically receive newly published packages. Instead, a separate process has to be triggered manually to update the packages in that repository. This completes the release cycle.

LESSONS LEARNED

It turned out to be good to have one single source of truth for templates, configuration files, etc. for our tools and to invest some time into automation. Thereby, files can be automatically updated in all projects when a new merge requests is opened, so that new conventions can easily be propagated. Also having a dedicated testing project turned out to be helpful to test changes of the CI configurations.

Firstly, building own Docker images for the CI saves requests to Docker Hub and, makes a big difference in CI pipeline run time. Secondly, the images come with the correct versions of the checking tools, which have to be kept synchronised with the local checks (*pre-commit*) to avoid frustration of code contributers and error tracking in the CI pipelines. Therefore, version pinning of the tools is necessary and can also be checked automatically.

Though we only enforce API documentation, investing additional time in writing easy accessible examples and "how-to-use" documentation increases the user-friendliness and adoption rate, eventually leading to a re-usable and sustainable code base.

Last but not least, it makes sense to automate the release process to enable new releases often, especially when new projects with many new features and more mature packages with minor fixes co-exist.

SUMMARY

We developed a set of Python packages to provide consistent interfaces to the KIT accelerators and facility infrastructure, making it useful for all levels of users—may it be for student projects or expert tasks in the control system itself. The libraries are developed and maintained by using a modern software development cycle in form of continuous integration with automated code checks, tests, and package releases. This makes the code maintainable, extendable and easy to use for all stakeholders.
We adopted GitLab, because it provides a single platform of this complex setup without the need to manage and maintain many individual solutions.

ACKNOWLEDGEMENTS

We thank E. Bründermann proof reading and value comments and recommendations.

REFERENCES

[1] M. J. Nasse *et al.*, "FLUTE: A versatile linac-based THz source", *Rev. Sci. Instrum.*, vol. 84, p. 022705, 2013. doi:10.1063/1.4790431

[2] A. Papash, E. Bründermann, and A.-S. Müller, "An Optimized Lattice for a very Large Acceptance Compact Storage Ring", in *Proc. IPAC'17*, Copenhagen, Denmark, May 2017, pp. 1402–1405. doi:10.18429/JACoW-IPAC2017-TUPAB037

[3] SoftIOC, https://github.com/dls-controls/pythonSoftIOC

[4] black https://github.com/psf/black

[5] *pytest* https://pytest.org

[6] *pre-commit* https://pre-commit.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP02

CONTROL SYSTEM FOR HESEB BEAMLINE AT SESAME

A. Abbadi*, A. Al-Dalleh, A. Aljadaa, M. Abugharbiyeh, M. Alzubi,
M. Genisel, R. Khrais, S. Matalgah, Y. Momani
SESAME Synchrotron-light, Allan, Jordan

Abstract

The HElmholtz-SEsame Beamline (HESEB) is a state-of-the-art soft X-ray beamline donated by Helmholtz research centre that was successfully installed and commissioned recently at Synchrotron-light for Experimental Science and Applications in the Middle East (SESAME). The control system design and implementation, which includes controlling low-level up to most sophisticated devices, has been done by SESAME's control engineers. This paper describes the design and implementation of the control system required to deliver the complete functioning of the beamline as well as the safety system including its measures put in place to protect the beamline's equipment and users.

INTRODUCTION

HESEB is the first soft X-ray beamline at SESAME and will significantly expand the research capabilities available to the user community in the Middle East and neighbouring regions. The source of the HESEB beamline is based on an elliptical polarising insertion device of the APPLE-type. The undulator's ability to provide linearly to circularly polarized light makes the beamline very suitable for materials science applications, especially magnetic materials. Its plane grating monochromator uses exchangeable gratings to cover a photon energy range from 70 eV to 2000 eV [1].

SESAME employs the Experimental Physics and Industrial Control System (EPICS) toolkit for controlling both machine and beamlines. EPICS is a control system middleware or framework for acquiring and controlling equipment through standardized communication between distributed components of different subsystems. The goal of beamline control system is to provide the users an easy interface to interact with the beamline components and devices. All beamline devices are connected to central units called EPICS Input Output Controllers (IOCs). These IOCs are hosted on Virtual Machines (VMs), there are three VMs per beamline, all of which reach the same network with the rest of the beamline's devices and hardware. A beamline network is separated from the other beamlines and machine networks. External access to beamline IOCs is provided through EPICS gateway.

CONTROL SYSTEM

Beamline control system consists of several sub-systems in which are illustrated in Fig. 1.

Equipment Protection System (EPS)

Each beamline has its own independent EPS in order to keep the devices and components safe while they are operating. It constantly checks the status of the beamline's parts and collects signals from field sensors to allow or inhibit the operation of beamline. This includes:

- Temperature measurement to protect beamline components where high heat loads are expected.
- Monitoring of coolant flow rates and react in case of insufficiency.
- Control the vacuum valves based on the signals received from the vacuum gauges and ion pumps.
- Watch the shutters' opening and closing time and report it to the control system. Also, it reacts in case of fail to close or limit switch failure.

PLC The EPS of SESAME's beamlines are PLC based. SIEMENS S7-1500 is the main controller used in HESEB. This model replaced the older SIEMENS S7-300, which was used in all beamlines before HESEB. Recently, a decision was made to consider the S7-1500 and S7-1200 in all new projects as the S7-300 model will be discontinued in 2023 [2]. Building of HESEB PLC control racks including PLC wiring, wiring diagrams, PLC coding and system commissioning have been done completely in-house by the control team and trainees.

PLC Programming The process of PLC programming begins with requirements analysis. Next, a matrix of causes and effects is defined, and finally, the PLC code is created and tested using this matrix. The PLC code structure is based on main routine, sub-routines and functions. This structure helps standardize the PLC programs and minimize programming errors. Moreover, it is easy to commission and understand, and flexible to upgrades.

EPICS Driver To connect the S7-1500 PLCs with EPICS, we used s7-nodave device support which is based on Asynchronous EPICS driver (ASYN) and libnodave library to communicate with S7 PLCs. s7-nodave does not require any special programming on the PLC side. Instead, the EPICS records just specify the memory address in the PLC to read or write the data [4].

Vacuum System

The control of vacuum system in HESEB consists of gate valves, TPG366 and TPG500 gauge controllers, IPCMini ion-pump controllers from Agilent and VAT fast valve controller. Remote monitoring and control over these devices is essential during operation of HESEB. The control interface

* anas.abbadi@sesame.org.jo

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP02

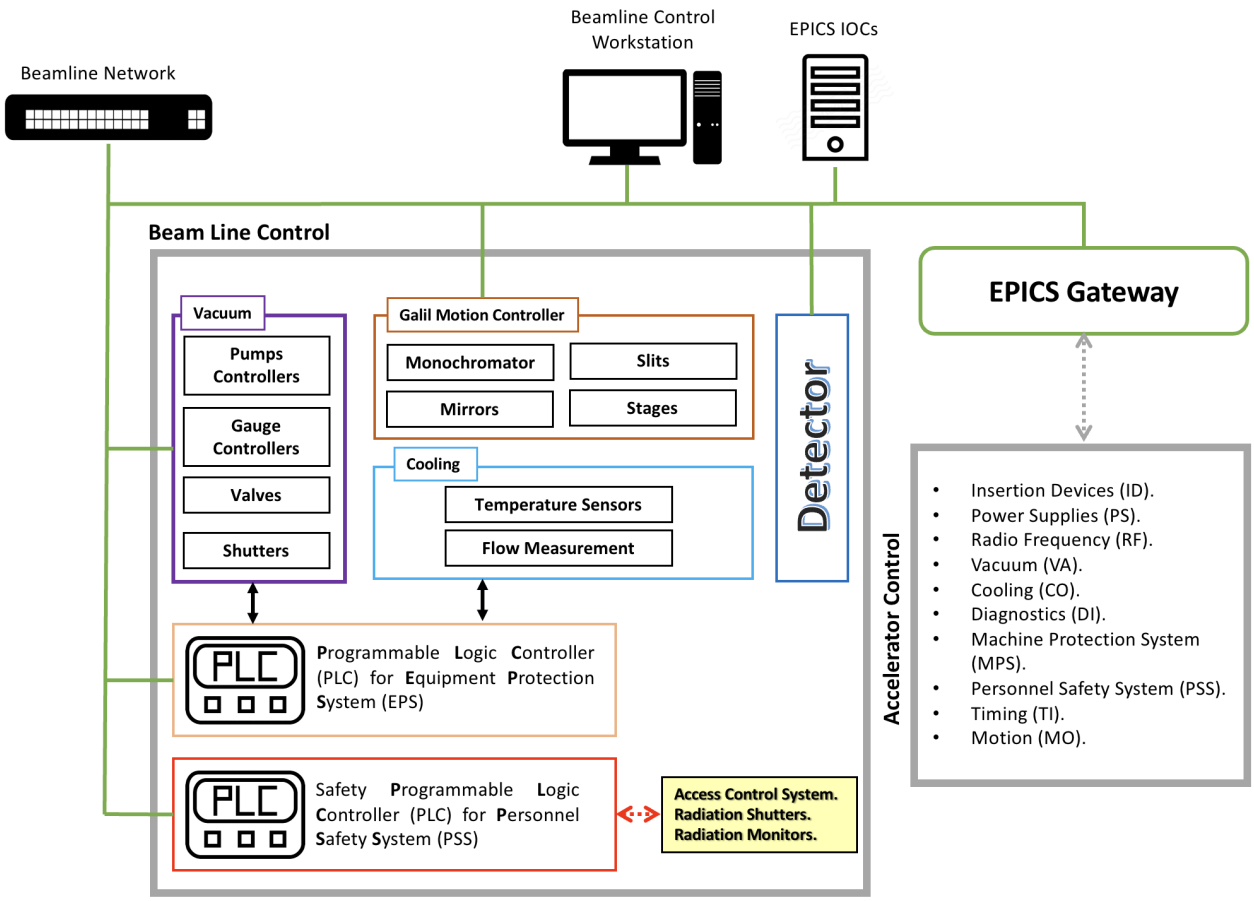


Figure 1: General layout of beamline control system.

can be divided into two categories; low level signal integration and smart controllers interface; low level signals are mainly the interlocks signals from gauges and pumps. These signals are transmitted to the PLC which will act accordingly to allow opening the valves or force them to close in case of emergency. Interlock levels can be configured either directly from the gauge/ion pump controller or from the control system Graphical User Interface (GUI). Gauge and ion-pump controllers are connected with vacuum IOC over Ethernet connections. A dedicated GUI shows the important parameters from gauge controllers; channel pressure, status, temperature, operating hours, in addition to read/write privilege over filter time, upper threshold, and lower threshold. Another dedicated GUI for ion pumps shows pressure level, current, voltage, maximum allowed power, pressure set-points, and current protection.

Motion System

SESAME uses Galil DMC as standard motion controllers. HESEB's motion system has four controllers that control the stepper motors of the beamline, namely for movable mask, slits, M1 mirror, Plane Grating Monochromator (PGM), M2 mirror, M3 mirror, M4 mirror, and sample manipulator. We use the standard ASYN based EPICS driver for Galil products, this driver is maintained and supported by the EPICS community. Mirrors and PGM coordinate motion were implemented using the EPICS standard transform record. The transform record supports multiple equations in a single record, however due to the complexities of the PGM coordinate motion we decided that each equation for motion would be implemented into one transform record. This allowed us to easily debug and trace the behaviour of the PGM's coordinate motion and prepare it for commissioning. Before moving the PGM motors, the EPICS motor simulation module (MotorSim) was used to test and validate the system.

Detectors

Three detectors are installed at HESEB, Bruker QM 100 and two Kiethly picoammeters. The QM100 is an X-Ray spectrum detector used for energy-dispersive X-Ray applications. For the control system, the device is supported by the Multi-channel Analyzer (MCA) EPICS module under Windows OS. On the other hand, ASYN driver was used to control and acquire current readouts from the Kiethly detectors.

Personnel Safety System (PSS)

As a soft x-ray beamline, HESEB does not require shielding hutches. Front end shutters and radiation monitors are connected to the main personnel safety system of the accelerator.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP02

HESEB CONTROL GUI

Beamline users interact with control systems through GUIs. SESAME beamlines, including HESEB, employ Qt as its GUI client. EPICS Qt is a Qt interface to EPICS' channel access protocol developed at the Australian Synchrotron [4]. The interface contains an API to read and write EPICS PVs across many widgets.

One of the main advantages of Qt is rich coding in GUI design; it allows implementing custom logic in the GUI. At SESAME, we run the Qt clients on a custom version of EPICS Qt where we customized some widgets like Indicators, labels and text boxes. This has been done by adding some channel access (CA) initialization and status update code. The main GUI in the beamline shows an overview of the beamline consisting of its three main sub-systems; front-end, optics and end-station as well as information from the machine through the EPICS gateway like beam current, energy, lifetime and ID gap. Block diagrams representing the beamline components are also displayed in the main GUI; the blocks are color-coded as follows:

- Red: component is interlocked
- Yellow: valve or shutter is closed.
- Green: component is in the operational state, or valve/shutter is open.

In addition, each block will show values for any connected thermocouples as well as pressure readouts for any gauges or ion-pumps. The buttons on the main GUI will show red color to indicate that this sub-system has an interlocked component.

PLANE GRATING MONOCHROMATOR

HESEB beamline consists of a collimated PGM attached to an elliptical undulator. The mechanics of the grating chamber comprises of two optical elements, a plane pre-mirror, and a plane grating in which they rotate around separate axes.

Control Aspects

PGM can be operated in one of three modes: fix deflection, fix diffraction, or fix magnification magnitude. The three modes of operation include set of equations, each mode has its own equations which must be enabled only when the relevant mode is selected. Equations of the other two modes must be disabled in order not to interfere with the running one. The complexity of the equations is hidden from users by an easy to use and interactive GUI. The design of PGM control software started by modeling the system graphically and building publisher-subscriber trees, Fig. 2 shows an example, then implementing the equations inside EPICS database using transform record, which provides the capability of solving set of equations sequentially. This technique provides an easy way to discuss the details of application with optics experts, it helps in software development, and it makes software testing easier.

Energy selection in PGM is done by selecting the proper angles: incident, deflection, and diffraction, the system also

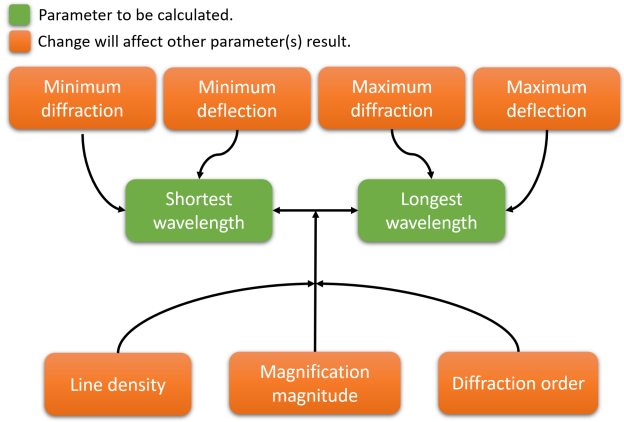


Figure 2: Publisher-subscriber tree for wavelength range calculation in fix magnification mode.

gives the user control over diffraction order, line density per meter, exit slit size and magnification magnitude – in fix magnification magnitude mode. Control system provides the user real-time information about energy resolution for the whole system and more details regarding each part (slit, pre-mirror, grating, etc.). Moreover, the system gives the user the ability to provide a range of deflection and diffraction angles and get the shortest and longest wavelengths.

PGM Safety

An important aspect in PGM control is safety. A set of safety layers to protect the mirror and the grating have been implemented. As a first layer of protection, control software calculates the target angles and triggers a stop command to the motors in case the calculated angle difference between them is below a certain threshold. Indicators in the GUI will turn red to notify the user to check the input parameters again.

Then the second layer of security comes in; if the angle difference touches the predefined threshold, in such scenario the motors are completely disabled by a hardware signal and it is mechanically latched, then the user cannot move the motors anymore until an expert personnel enables them again manually.

REFERENCES

[1] W. Drube, MF. Genisel, and A. Lausi, "SESAME Gets Soft X-Ray Beamline HESEB", *Synchrotron Radiat. News*, vol. 35 - 1, pp. 22-22, 2022. doi:10.1080/08940886.2022.2043710

[2] SIEMENS, https://www.siemens.com/

[3] S. Marsching, "A New EPICS Device Support for S7 PLCs", in *Proc. ICALEPCS'13*, San Francisco, USA, Oct. 2013, paper THPPC027, pp. 1147-1149.

[4] A. Rhyder, R. N. Fernandes, and A. Starritt, "Qt Based GUI System for EPICS Control Systems", in *Proc. PCaPAC'12*, Kolkata, India, Dec. 2012 paper WECC03, pp. 10–11.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP04

EPICS TANGO BRIDGE

T. Madej, P. Goryl, M. Nabywaniec, Ł. Żytniak, S2Innovation, Kraków, Poland

Abstract

EPICS (Experimental Physics and Industrial Control System) [1] and Tango Controls [2] are the two most popular control systems for scientific facilities. They both have advantages and disadvantages. Sometimes there is existing software driver supporting integration only with EPICS or Tango. EPICS Tango Bridge is the perfect solution for accessing Tango devices using EPICS Channel Access protocol. Using our bridge, the cost of integration is significantly lower than providing dedicated integration of specific hardware in EPICS.

The bridge provides high reliability and robustness. Test cases created during the development process verify their limitations like the response time of reading one attribute in the bridge with a different number of Process Variables (PVs) or parallel access for multiple EPICS clients and many others.

OVERVIEW

Architecture

EPICS Tango Bridge provides access to Tango servers via the EPICS Access Channel. The architecture of the tool is shown in Fig. 1. We used the PyTango [3] library as the interface of Tango devices servers.

The project is based on the PCASpy library, which provides biding to EPICS Channel Access. The main components of EPICS Tango Bridge are pcaspy's SimpleServer and TangoDriver which are inherited from pcaspy's Driver class.

SimpleServer creates PVs based on the PVDB file and runs IOC (Input Output Controller). The server encapsulates transactions performed by the channel access server.

Driver Class is the base class that connects channel access requests to a real-world data source. The base class implementation simply stores a user-written value and retrieves it on request. TangoDriver, extending Driver class allows to read and write attributes of Tango Device Server.

TangoDriver during the initialization creates multiple Manager objects, which provide access to attributes and commands of specific Tango Device Server using the PyTango DeviceProxy class. There are multiple implementations of the Manager interface according to the specific usage (e.g., polled attributes, commands).

Each Manager object creates its DriverAdapter thanks to whom, write to PV (Process Variables) causes an update of value in Tango Device Server's attribute.

Usage

To run the bridge, we need to pass the corresponding PVDB file, providing a valid mapping between EPICS PV and Tango attributes and commands. That file contains a simple Python dictionary where the keys are the base names of the PVs, and the values are their configurations. TangoDriver creates multiple managers providing access to Tango Device attributes. Each manager implements the read and writes method. The write method is realized using the write thread. In that thread, the new value is assigned to the Tango Device Server attribute and then to the corresponding PV using the DriverAdapter object. If the Tango attribute is a two-dimensional list, it will be flattened during reading from the PV, i.e. [[1,2], [3,4]] will be read as [1,2,3,4]. During writing to the PV, the Bridge IOC will attempt to restore the correct structure before passing the data to Tango.

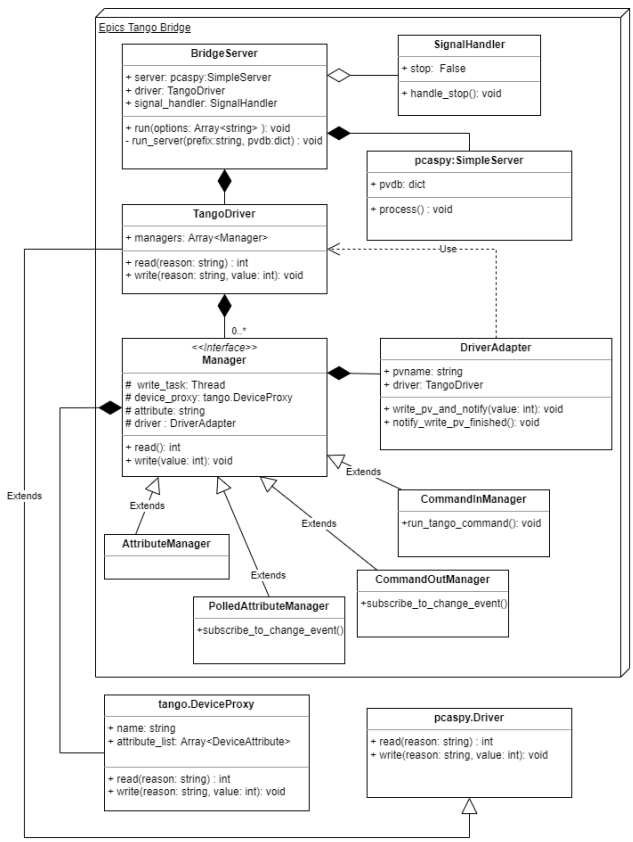


Figure 1: EPICS Tango Bridge Architecture.

STRESS TEST

To recognize EPICS Tango Bridge's capabilities and limitations, we prepared a set of stress tests.

In this section, we will describe the test scenarios as well as the test setup.

Test Scenarios

To test the performance of EPICS Tango Bridge, we used the PyTest [4] library. We prepared the following test scenarios:

1. Key Performance Indicators (KPIs) test – checking the maximum size of PVDB databases due to the type of attribute (double, spectrum double, string). Tests

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP04

check the correctness of EPICS Tango Bridge operation in two ways using EPICS and Tango Controls commands.

2. Read/write time test – checking the time of read and write using PyTango and PyEpics [5] access. As parameters, we pass number of PVs which are run in Bridge IOC and the type of attribute and scan option. During the test, firstly proper PVDB file and example PyTango Device Server are generated. Then Tango Server and Bridge run the test we compute the average time of the read and write.
3. Read time during simultaneous access by multiple clients to one Bridge IOC. In that scenario, one client performs multiple writing actions to PV using the PyEpics caput command. Multiple clients read from the corresponding Tango Device Server attribute. PVDB file contains one Double Scalar attribute. As a parameter for the test, we pass the number of reading clients (default 20), and the number of write and read actions (default 200). As a result, we compute the average time of read action. During the test we validate the correctness of read attributes as the writing client writes a sequence of some numbers – every client has to read the same sequence.

The tests were conducted on a local virtual machine running Ubuntu 18.04 (64bit, x86, HVM) operating system. Following software was installed in the virtual machine: PyTango 9.3.3, PyEpics 3.5.1, PCASpy 0.7.3. Test 3 was conducted using Docker containers.

Test Results

Table 1 shows us how the reading time varies with different sizes of PVDB files performed for the double and a twenty-character string attribute. Each attribute reading was repeated one hundred times and the meantime and standard deviation were calculated for them. We can observe that the reading time is similar for both clients.

Table 1: Read Time for EPICS and Tango Tested in Python Client

Size PVDB	Type Attribute	Average TANGO read time [ms]	Average EPICS read time [ms]
1	double	0.50 ± 0.13	0.58 ± 0.82
10	double	0.50 ± 0.12	0.65 ± 0.85
100	double	0.52 ± 0.14	0.54 ± 0.75
1000	double	0.50 ± 0.14	0.40 ± 0.42
1	String	0.48 ± 0.12	0.52 ± 0.67
10	String	0.51 ± 0.11	0.66 ± 1.01
100	String	0.51 ± 0.13	0.68 ± 0.96
1000	String	0.52 ± 0.14	0.61 ± 0.76

Table 2 shows us an analogous situation as above, in this case, we check the attribute write time also using the Tango Controls and EPICS client python commands. The time of writing the attribute using the EPICS client is longer, despite all this time being satisfactory.

Table 2: Read Time for EPICS and Tango Tested in Python Client

Size PVDB	Type Attribute	Average TANGO write time [ms]	Average EPICS write time [ms]
1	double	0.23 ± 0.14	102 ± 3
10	double	0.24 ± 0.15	103 ± 6
100	double	0.23 ± 0.15	112 ± 11
1000	double	0.15 ± 0.11	112 ± 11
1	String	0.21 ± 0.14	103 ± 2
10	String	0.23 ± 0.13	105 ± 6
100	String	0.06 ± 0.04	112 ± 12
1000	String	0.14 ± 0.09	111 ± 18

During test 3, we ran twenty clients using docker and all of them were doing operations of reading, one client wrote randomly generated values, and we checked the average read time for one of the reading clients and the result was about 0.52 ± 0.24 ms.

CONCLUSION

EPICS TANGO BRIDGE is a tool for accessing Tango Classes via EPICS Channel Access, which has scalar attribute support (normal and polled), command execution, and support for spectrum (one-dimensional array) and images (two-dimensional array). Advantage of the bridge is its ease of use, it is enough to prepare a PVDB file before use, which is a dictionary linking the attribute name with a specific PV structure.

Efficiency and performance are important aspects of control systems. The bridge has no significant impact on the speed of reading, with a bigger number of PVs in the PVDB file or with more clients working on the same server, the time is quite the same and is on the order of a fraction of a millisecond. Only write time using caput in relation to Tango commands is the time is relatively longer.

REFERENCES

[1] EPICS, https://epics-controls.org

[2] Tango Controls, https://www.tango-controls.org

[3] PyTango, https://pytango.readthedocs.io/en/stable/

[4] PyTest, https://readthedocs.org/projects/pytest/

[5] PyEpics, https://github.com/pyepics/pyepics

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP05

INTEGRATION OF QUENCH DETECTION SOLUTION INTO FAIR'S FESA CONTROL SYSTEM

Matic Marn, Andrej Debenjak, Cosylab d.d., Ljubljana, Slovenia
Michal Dziewiecki, GSI Helmholtzzentrum für Schwerionenforschung, Darmstadt, Germany

Abstract

Facility for Antiproton and Ion Research (FAIR) is going to make wide use of superconducting magnets for its components: the SIS100 synchrotron, the Superconducting Fragment Separator (SFRS) and Atomic, Plasma Physics and Applications (APPA) experiments. For all these magnets, uniform quench detection (QuD) electronics have been developed to protect them in case of uncontrolled loss of superconductivity. The QuD system will contain ca. 1500 electronic units, each having an Ethernet interface for controls, monitoring, data acquisition, and time synchronization. The units will be grouped into sub-networks of ca. 100 units and interfaced via dedicated control computers to the accelerator network. The interfacing software used to expose QuD functions to the FAIR controls framework is implemented as a Front-End Software Architecture (FESA) class. The software provides a solution for the constant collection of the data and monitoring of the system, storing the complete snapshot in the case a quench event is detected, and prompt notification of a quench to other components of the FAIR facility. The software is developed with special attention to robustness and reliability.

INTRODUCTION

Quench (uncontrolled superconductivity loss) events are considered a major threat in superconductor technology. During a quench of a superconducting magnet, the energy stored in the magnetic field is rapidly released in the quench area, which may cause severe damage. However, damage can be avoided by quench protection devices like quench heaters or dumping resistors. To activate them right in time, QuD systems are used.

FAIR's SIS100 synchrotron, the SFRS and APPA will employ a number of superconducting magnets for their operation [1]. For these magnets, voltage-measurement-based QuD methods are utilized: direct voltage measurement for current leads, bridge measurement for high-current magnets and pickup coils for low-current magnets [2]. Voltages over all superconducting components are monitored, mostly redundantly, by assigned QuD units. Each unit consists of a custom analog front-end (specific for each monitored part type), a set of comparators (quench detection relies on comparing acquired signals against pre-programmed thresholds), and circuitry for device control and data acquisition.

The data acquisition feature is not involved in quench detection itself, but collecting signals allows in-depth analysis of magnet operation and 'post-mortem' data are crucial for understanding magnet failures once they occur.

FAIR's control system is FESA-based, and the integration of the QuD units is done via a FESA class. The QuD FESA class was designed to control up to 96 QuD units by a single equipment class, however, there is no technical limitation to extend this number further as far as enough processing and memory resources are allocated to the hosting computer.

FESA is a C++ framework used for developing real-time (RT) software for devices that have to be integrated into a control system – software that controls and monitors accelerator equipment and performs data acquisition. It comes with a rich environment for designing, developing, deploying, and testing. It offers wide possibilities in data acquisition, data handling, and standardized generic interfaces called properties, which can be specified for each device being integrated. An acquisition property allows for acquired data to be published (i.e., notified) to the users, while command and setting properties facilitate the user to provide input to a FESA class. In addition, FESA provides tight and seamless integration of the FAIR's White Rabbit [3] timing system and focuses on real-time execution.

FAIR QUENCH DETECTION UNITS

Each QuD unit can be divided into two parts: the analog QuD circuitry (which is virtually independent of the control system or any other high-level electronics) and a module for device control and data acquisition. The latter is built around a Cyclone IV FPGA and runs a NIOS-II soft-core with FreeRTOS-based software. Its functions range from simple operations (like remote forcing or resetting of the quench signal) through controlling quench thresholds to continuous 10kHz signal acquisition. Further, it provides numerous diagnostics features and remote updates.

Digital Interface

All units are controlled via a fast Ethernet interface running multiple protocols on top of IP v.4. For generic device control, the Simplified Universal Serial Interface (SUSI) protocol [4] has been developed, which utilizes a variable size register model for device programming. For data download, a custom real-time data transfer protocol has been implemented directly in FPGA hardware to offload the softcore. Further, some generic protocols (DHCP, mDNS) are used to support IP connectivity.

It's worth noting that the quench signal is transmitted over a dedicated digital line and it's independent of the network interface.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP05

Data Format

QuD units sample 8 channels of analog data with a nominal frequency of 10 kHz along with further 16 binary status signals. These are packed into frames consisting of 64 samples and a summary. The summary contains mean values of analog channels, a snapshot of the binary status, measured quench threshold values, frame number, and last but not least, its timestamp. Each frame counts 1328 bytes which fit into a single UDP frame, allowing easy and efficient data transmission.

Time Synchronization

Local data timestamping is used to synchronize streams coming from QuD units. Timestamps are added to data at the earliest possible stage in the unit hardware rather than in the global data aggregator. This way effects of random delays on the network are eliminated. To provide this, all units must be equipped with local clocks and they must be synchronized to an external global clock. The synchronization is achieved utilizing the Precision Time Protocol (PTP).

Internally, units use sample and frame triggers to control the data acquisition. These triggers can be free-running or synchronized to the global clock. In the latter case, frames coming from all synchronized units will have the same timestamps, which makes data aggregation much easier.

SYSTEM OUTLINE

Hardware Arrangement

QuD units are logically (and mostly also physically) grouped into subsets of up to 96 units with a single control server. The local network between the server and units is isolated from the general accelerator network so that direct access to units from outside is not possible. Each server runs an instance of a dedicated FESA class as described below to provide QuD control and data acquisition.

FESA Architecture

To integrate the QuD units into the FAIR control system a FESA class application was designed. The class supports real-time data acquisition from all the units which stream the data by User Datagram Protocol (UDP) packets. The class periodically monitors the units and subscribes or unsubscribes from the unit data streams based on the operational status of each unit. This design allows robust and reliable data availability. The data must be made available to any FESA client for a certain amount of time after it has been acquired. For that reason, once the data is received it is put inside a First-In, First-Out (FIFO) data acquisition (DAQ) buffer which supports a single-write-multiple-read interface with minimal locking required for synchronization. The data is available on demand as either raw frame data or processed scaled data until it is overwritten in the buffer. The data is also made available to the interested clients continuously as it is received at a 10 Hz update rate. The class monitors received data and as soon as a quench event is detected on one of the units it signals an interlock to the FAIR interlock system. In addition to the DAQ capabilities, the QuD FESA class also provides command and setting properties for the user to setup and configure the QuD units.

Even though each QuD FESA class instance interacts with up to 96 QuD units, the FESA class only implements one software device-instance. While from the FESA perspective it would be most appropriate that one device-instance represents one QuD unit, this is not the case for this particular implementation. With the complete QuD system containing around 1500 QuD units and each unit being represented by a corresponding FESA device instance, this would introduce a large amount of FESA software devices to the control network which is inconvenient and unbefitting.

IMPLEMENTATION DETAILS

DAQ Buffer

The QuD FESA class receives the real-time data from up to 96 QuD units. The data needs to be aggregated, aligned, and stored for a predefined period (typically several minutes). The rate of received data is substantial and was one of the main aspects affecting the overall design of the buffer.

The whole buffer is pre-allocated in advance to avoid expensive memory allocations and moves during run-time. Memory for individual units is located in a continuous fashion to increase locality and reduce the number of cache misses.

Data from different units are time-aligned by the timestamp of the frames. For the frame to be inserted into the buffer the timestamp has to match exactly. This means all the units need to be configured to use the global clock as a trigger reference.

Performance tests have shown that a single writer thread can easily handle the expected load so the buffer is designed to be used by a single writer thread. This way, the synchronization between multiple writers is avoided.

The buffer is split into sections called blocks. Each block consists of multiple DAQ frames. At any time, the data may be written into the buffer only in the sliding writer window which is 2-block wide. When the data is written one block past the writing window the window slides by one block forward to include the new block. The old block which is no longer included in the writing window may not be written to anymore and may be read from.

Reader threads do not block the writer thread while they are reading the data. This is achieved by the reader threads reading the current position of the writing window before and after the data was read. With that, the reader thread can determine if the data was overwritten while being read. Reading and updating the writer window position is synchronized with a mutex lock, which affects the performance negligibly.

Communication Sockets

The QuD FESA class implements two types of sockets to communicate with units. First, a general purpose SUSI socket is used to write to and read from a QuD unit. The

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP05

communication is entirely done using UDP packets with two-way communication for each interaction. Appropriate synchronization mutex is used to make sure that each interaction is correctly handled. The second implemented socket is a real-time socket which is only used to receive the streamed real-time data. It exposes the Linux socket file-descriptor which allows to efficiently poll for new data for all units.

Data Acquisition

A FESA custom event source is used to receive the streamed real-time data. When an acquisition start command is sent to the QuD units, they start streaming UDP packets containing the real-time data. Due to the relatively large number of units and to improve the performance, the Linux poll command is used. Whenever one of the sockets has available data, the data is read, deserialized to a DAQ frame structure, and put inside the DAQ buffer. As soon as a new block is written into the DAQ buffer, the custom event source triggers the acquisition RT action. Before adding the frame to the buffer, it is also checked whether a quench status flag is set. If that is the case, the quench handling RT action is immediately triggered. The custom event source thread only does the absolutely most critical work required and other less time-sensitive operations are handled in other lower-priority threads.

Data Processing

Some data acquisition modes require the data to be processed before being sent to the interested clients. There are two processing procedures implemented. Namely, the data may be decimated and scaled. For decimation, a user-specified number of data samples is averaged. For scaling, the data can be scaled from raw ADC values to volts according to preset calibration coefficients.

Data Acquisition Modes

QuD data is acquired in the form of raw DAQ frames, deserialized, and stored in the DAQ buffer. It resides in the buffer until it is requested by the user or a continuous notification is triggered. Three modes of publishing the data, are supported i.e., *Raw*, *Continuous* and *Post-mortem*.

Raw UDP packet data (bytes), can be requested with *GetRawData* command FESA property. The user has to supply the timestamp of the first frame of interest and the number of frames required. As long as the requested data is available in the DAQ buffer, it is serialized back to raw bytes and sent to the user.

In addition to the raw data, the user can subscribe to continuous data notifications. This allows getting a continuous stream at 10 Hz update rate. The data is decimated and scaled to Volts before sending to the user.

The last available data acquisition mode is post-mortem acquisition. Since a quench event may be triggered at any time, it is paramount for the post-mortem analysis to have access to the acquired data before and after the quench happened. In this case, the user may request the post-mortem data with the *GetPostMortemData* command property. The user supplies the timestamp of the first frame of interest and the number of frames. The data is read from the DAQ buffer, scaled, and sent to the user.

Robustness and Reliability

A single class instance works with multiple units. One or more malfunctioning units should not degrade the performance of the class or the ability to use other units.

To achieve the desired reliability, a health monitoring system is implemented. The QuD FESA class maintains internal statistics of the received data frames. With each received packet the statistics are updated. These statistics are then periodically monitored and appropriate action is taken for each unit in case of detected misbehavior. For example, when a unit is correctly configured and is not sending data when it should, the acquisition is restarted on the unit. Similarly, when the unit is misconfigured and is sending data, the acquisition is stopped on the unit, since the sent data is considered invalid. In addition, periodical checks are implemented that units are correctly synchronized and configured at all times.

Each unit can be masked (i.e., easily excluded from the system), in case there is some issue with the unit that cannot be fixed remotely and physical interaction is required. When a unit is masked it effectively means that it is ignored by the QuD FESA class and as such the unit is not operational.

Standard DAQ API

The class exposes the acquired data via the FAIR standardized DAQ Application Programming Interface (API). This allows a common client Graphical User Interface (GUI) implementation to be used across multiple FAIR accelerator control solutions.

CONCLUSION

QuD units are an essential part of preventing damage to the accelerator devices in case of a quench event. As such they are integrated into the FESA control system. Care is taken to provide a performant, robust and reliable solution. At the same time, the implementation offers the user full access to the settings and configuration of the QuD units. Data acquired from the QuD units is buffered, processed, and made available to interested FESA clients.

REFERENCES

[1] W. F. Henning, "FAIR - an International Accelerator Facility for Research With Ions and Antiprotons", in *Proc. EPAC'04*, Lucerne, Switzerland, paper TUXCH02, pp. 50–53, July 2004.

[2] P. Szwanguber *et al.*, "Study on Mutual-Inductance-Based Quench Detector Dedicated to Corrector Magnets of SIS100", *IEEE Trans. Appl. Supercond.*, vol. 29, nr. 4, pp. 1-5, 2019. doi:10.1109/TASC.2019.2896139

[3] *The White Rabbit Project - official web page*, August 2022. https://white-rabbit.web.cern.ch

[4] M. Dziewiecki, "Quench Detection System Developer's Handbook", Internal GSI document, Darmstadt, 2022.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP07

A MODERN C++ MULTIPOCESSING DOOCS CLIENT LIBRARY IMPLEMENTATION

S. Meykopff†, Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract

At the DESY site in Hamburg/Germany the linear accelerators FLASH and European XFEL are successful operated by the control system DOOCS. DOOCS based on the client–server model and communicates with the matured SUN-RPC. The servers are built with a framework which consists of several C++ libraries. The clients use a DOOCS client library implementation in C++ or Java. In the past years the public interface (API) of the C++ client library was refined. But modern C++ features like futures are not provided in the API. Massive multi-processing, parallel communication, and optimized names resolution could improve the overall communication latency. The usage of the standard C++ library, the limit of external dependencies to ONC-RPC (former SUN-RPC) and OpenLDAP, and the reduction of the code size, increases the maintainability of the code. This contribution presents an experimental new client C++ library which achieves these goals.

BIRDS EYE ON DOOCS

The DOOCS control system uses the proven ONC-RPC (formerly SUN-RPC) for data transmission. To establish a connection to an ONC-RPC server, you need a host name and a port number. A DOOCS address must therefore first be resolved into a combination of host name and port number. This resolution is started via an LDAP query. If the server locations are served via multiple servers, all required information is available in the LDAP response. If all locations are answered via a single server, the existing locations can be queried via an RPC call to these servers.

Address Resolving with LDAP

The LDAP connection is established with OpenLDAP. First a connection to the LDAP server must be bound via ldap_simple_bind_s(). The single queries are then made with ldap_search_ext_s(). Depending on which entries are searched for, the distinguished name (DN) must be created at runtime. The DN consists of the following Relative Distinguished Names (RDN) "location", "device", "facility", "dc". The filter consists of a string with a "server-mask" and partly "device" as logical link. The last parameter to be specified is the scope in which to search. This also depends on which DOOCS address part is searched for. The connection to the LDAP server can be reused until all address resolutions are finished. At the latest at the end of the program the connection must be closed with ldap_unbind(). A sequence diagram is shown in Figure 1.

LDAP Answer

The LDAP query response contains these fields: "facility", "device", "location", "property", "server", "host", "channel", "protocol", "lib-prog", "server-mask", "status". The first four are part of the DOOCS address. The fields "host" and "lib-prog" are needed to connect to a DOOCS server. The two fields contain the above mentioned connect information for the RPC call. When searching for locations, the "property" field remains empty.

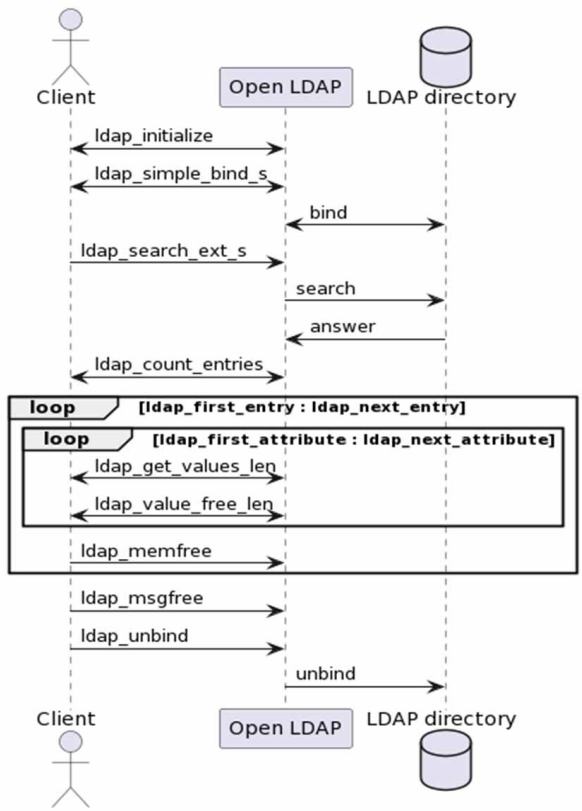


Figure 1: Address Resolving.

Authorization

To establish an RPC connection, authorization must first be performed. The authorization is provided by the authunix_create() function. The function requires the target host name, the (Unix) user ID and the group ID of the person to be logged in as parameters. The authorization is valid for all threads of a task until the auth_destroy() function deletes it again.

RPC Bind

A connection to the target server will be opened with clnt_create(). The function needs the host name as parameter and the port number determined above as parameter. The choice whether UDP or TCP is used as protocol must now be made. However, UDP is not used in DOOCS because the RPC messages may otherwise have a maximum of 8 kbytes of coded size. The function expects the protocol type as string. After the connection has been established,

† sascha.meykpoff@desy.de

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP07

the function returns a client pointer. The authorization created above must now be entered in the cl_auth field. From now on the connection can be used for communication with RPC messages.

Serialization with XDR

The core of RPC communication is a function call on another system. Not only must a function be started over the network, but function parameters or return value must also be transferred. These parameters must be serialized. Since the foreign system or also the transmission layer can use another byte order, the serialization must consider this also. ONC-RPC describes the parameters in its own language. An RPC specification (XDR) contains a number of definitions. The language supports scalar data types, arrays, constants, enumeration, struct, and union. The parameter description is normally stored in .x files. The program rpcgen creates from the .x files client and server stubs in C. To remain compatible to DOOCS the original DOOCS source files are used. The DOOCS library encapsulates the data in EqData objects. These simplify the copying of the data considerably. For simplicity, these EqData objects are also used for this project.

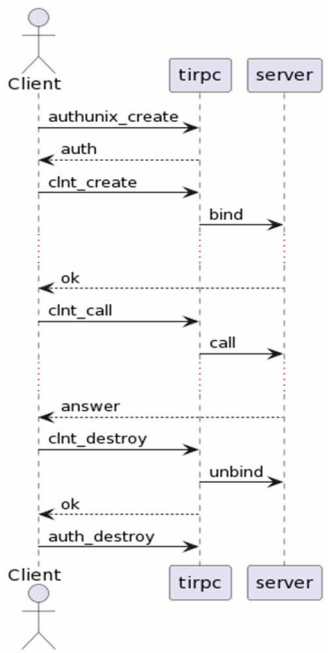


Figure 2: RPC Sequence Diagram.

Remote Call

To do a data exchange, the ONC-RPC function clnt_call() is called. The function expects the client handle, the proc number and a timeout. The proc number is an enumeration on get, set, or get_name. A pointer to a param_block structure describes the data to be sent. The structure consists of a pointer to an EqNameString structure and a pointer to an EqDataBlock structure. In the EqNameString the parts of the destination DOOCS address must be entered. The EqDataBlock structure is the fundamental DOOCS data container. This container consists beside some timestamps and error codes mainly of the DataUnion structure. The DataUnion structure has the field data_sel, which specifies the DOOCS data type. In the following union all DOOCS data types are described. How the union is evaluated depends on the data_sel field. Graphic 2 provides an overview.

Return Value and Cleanup

How the structure must be serialized is communicated to the clnt_call() function via a pointer to the data generated with rpcgen. The return value of the RPC call is written to an EqDataBlock structure which is also passed. This structure must also be described by a pointer on an RPC specification. The clnt_call() call is blocking. It returns a clnt_stat parameter as status, which is RPC_SUCCESS if successful. The EqDataBlock structure of the return parameter is allocated with malloc by the clnt_call() function. If the structure is no longer needed, it must be freed using clnt_freeres(). An unused connection is closed with clnt_destroy().

Receive Location Information

If you need to retrieve the list of locations or properties from a server, this is done via a modified RPC call. Instead of the proc number for get, the enumeration value for get_names is used. The location and the property name parts of the DOOCS address remain empty. The response from the server is a list of USTR data types. The USTR type contains among other things a string. Here the location name is stored.

Receive Location Information

In the following, a use case is considered where different control system values are read in simultaneously during a measurement. The classic DOOCS API makes several synchronous blocking RPC calls here. Here the addresses are resolved one after the other and then the blocking clnt_call is called in each case. A full sequence is shown in Fig. 2.

OPTIMIZATION

The first optimization concerns the API. If the data is returned directly in the API, it is also necessary to wait until the data has been transported completely. The experimental API returns only one std::future. In the API a thread serves the corresponding std::promise. If the data has arrived, the method set_value() of the promise is called. The method set_exception() is used in an error case. The client can wait using the wait() methods of the future until the data arrived. In the meantime, the client can start other requests, or do something else. If the wait function signal returns, the client can fetch the data with the get() method. In the best case the client waits now maximally for the time of the slowest call.

Parallel LDAP Calls

By simply replicating an LDAP database, several servers can easily be set up in parallel. Normally, these are backup servers that step in when one server is unavailable. An optimization is obvious at this point. The LDAP queries

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP07

put into a queue. A thread pool then processes the queries in parallel. A separate thread is used for each LDAP server. This allows multiple LDAP servers to be used simultaneously. A connection to the LDAP server should not be terminated immediately. Since it cannot be predicted when the next LDAP query will occur. A timer should terminate unused connections after a defined time.

Parallel RPC Calls

The RPC call can also run in parallel. In this case, communication with each server takes place via a separate thread. To conserve the resources of a client system, an upper limit for threads should be observed. In this case, a thread must process several connections. More than one connection to a server must be prevented at the moment, since it is not yet known whether the servers are internally multi-threaded. Also, the RPC connections are not closed after the last access. The slow connection setup must otherwise be restarted for future queries. A timer must also close unused connections after a certain time. A full sequence of two parallel calls is shown in Figure 3.

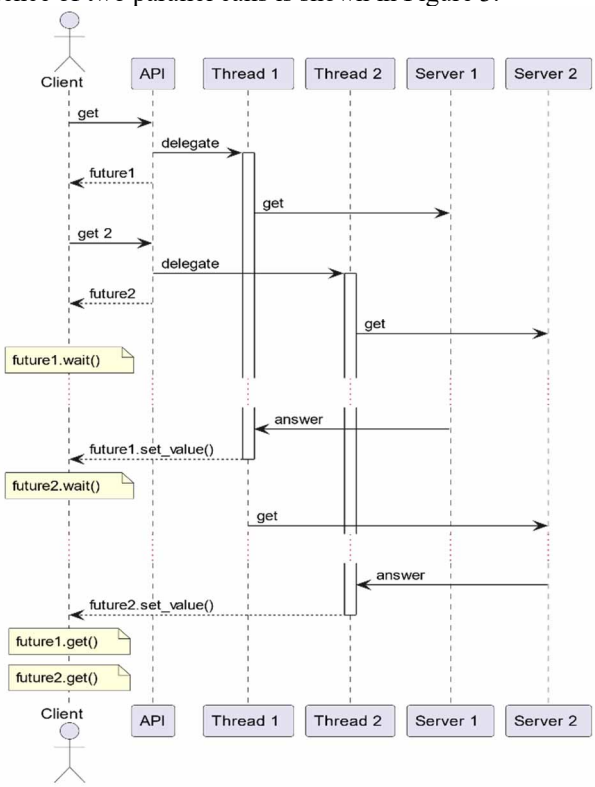


Figure 3: RPC Optimization.

IMPLEMENTATION

At the beginning, the API shall only support an asynchronous query of DOOCS properties. The method call in the RPC_API object is defined as follows:

```
std::future<std::unique_ptr<EqData>> read(
    const std::string &doocs_address,
    EqData *send_data
);
```

The doocs_address parameter contains the DOOCS address in the complete form ("FAC/DEV/LOC/PROP") and is separated internally into the individual address parts. The EqData parameter is a pointer. So that the passed data can be released or processed immediately after the end of the function, the EqData data block must be saved internally. The return value is a std::future with a std::unique_ptr on an EqData data block as described above. The scope of the internally generated EqData block is shifted to the user code by the unique_ptr. If the user does not need the block any more, it is released immediately. The proceeding should prevent memory holes. The use of EqData is purely a convenience decision. The EqData object provides easy access to all data types and also handles the copying of data.

RPC_INFO Object

If the API works with std::future, a std::promise must be managed internally. The std::promise must not be answered until the data request is finished. The promise is saved with other information such as the parts of the DOOCS address, data to send, timeout information, host server names and port numbers. On the first call in the experimental API the necessary information is copied into a rpc_info object. The rpc_info object can now be moved to the individual processing queues. At the end of the processing chain the promise is answered and the rpc_info object will be freed.

LDAP Cache

The already resolved LDAP requests are stored in a ldap_cache class. A std::unorderd_map holds the data and a mutex provides thread safety. Still within the get() method of the rpc_api class, the ldap_cache is searched for an entry. If none is available yet, the rpc_info object is put into the queue for LDAP query.

LDAP Queue

The ldap_queue is a std::queue which takes the rpc_info object. A thread pool get the data from the queue and starts the LDAP requests. Successful requests are recorded in the ldap_cache. For this purpose, a std::function object is defined in the rpc_info object. The function references the ldap_cache object and will be used as a callback. Thus the ldap_queue has no dependency to ldap_cache. Testing the ldap_queue is made easier by having fewer dependencies. If the LDAP query is not yet sufficient the server is queried via RPC. The rpc_info object is pushed into the RPC queue described later. Once the name resolution is complete, a second std::function is used as a callback back into rpc_api. In the scope of rpc_api, the RPC_INFO structure is moved into the rpc_scheduler.

RPC Scheduler

The dispatch() method of the rpc_scheduler object receives the rpc_info object. The rpc_scheduler has a rpc_dispatch object for each RPC connection. The rpc_dispatch object consists of a thread and a rpc_info queue. If no rpc_dispatch exists for the RPC connection, a new rpc_dispatch object is created. In the constructor of rpc_dispatch a rpc_auth object is necessary. Later the

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP07

rpc_auto object will authorize the communication. The dispatcher now moves the rpc_info data into the rpc_dispatch object. The thread of rpc_dispatch will take an entry from the queue and continue with the processing. For the RPC call a connection to the server must exist. Therefore rpc_dispatch has a rpc_connection_map object. The object stores each connection in an unordered_map. This is important when rpc_dispatch needs to make queries to multiple servers. A centralized map is not possible because in the current ONC-RPC implementation each thread must establish its own connection. The rpc_connection_map object now creates a server connection and also terminates it when the rpc_connection_map is cleaned up. The RPC handle can now be read from the rpc_connection_map object and the RPC call can be performed. Once this is finished, the res_block structure is transformed into an EqData object and copied into a std::unique_ptr<EqData>. This will be moved into the callback. The callback ends up in a method in the rpc_api object. The rpc_api scope moves the value into the std::promise. Now the rpc_info object is released and the query is finished.

MEASUREMENT

For the operation of the machines European XFEL and FLASH mainly JDDD is used. JDDD structures the data in so-called panels. The data read from the control system is displayed there. During a shift handover at the machines, screenshots are printed from the main panels into the electronic logbook. In total, there are about 30 screenshots. To get a valid collection of DOOCS addresses, these panels were opened simultaneously and the list of currently used DOOCS addresses were written to a file. The list contained about over 12000 entries. Duplicate entries, addresses of foreign control systems, one particularly slow server, and unreachable addresses were removed from the list, leaving just over 2800 usable DOOCS addresses pointing to 287 servers.

Timing Results

By getting the data from all addresses the runtime of the classic DOOCS library and the experimental library was measured. The results are shown in Table 1.

Table 1: Runtime Results

Code	Runtime	No. Threads
Classic	280s +-34s	1
Exp.	13s +- 1s	100
Exp.	13s +- 2s	50
Exp.	31s +- 8s	25
Exp.	64s +- 11s	10
Exp.	105s +- 12s	5
Exp.	143s +- 11s	2
Exp.	170s +- 21s	1

Figure 4 shows when the individual operations of the Classic DOOCS library are finished. About 200 queries require a total of 90s. A peak of 6 seconds (not plotted) for one call was measured. These slow responses are the lower limit for the experimental library.

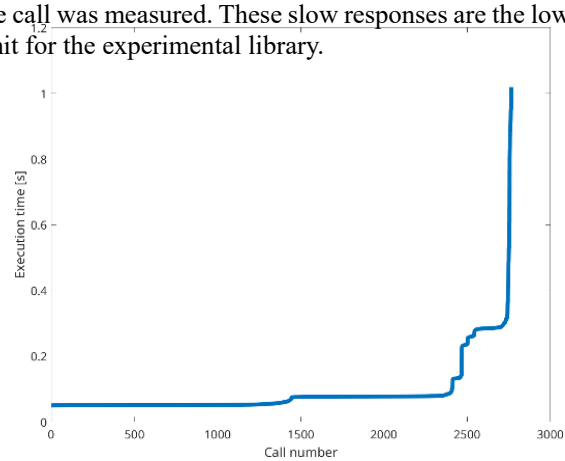


Figure 4: Runtime of finished RPCs.

Runtime Histogram

In the experimental library case Figure 5 shows a histogram when (since program start) how many queries have been completed. The most requests were finished after 10 seconds.

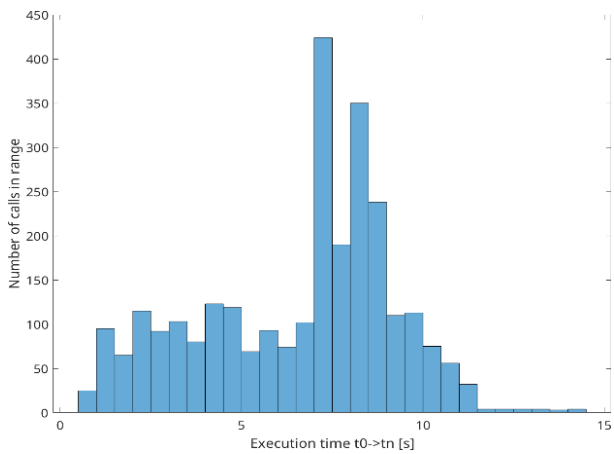


Figure 5: Runtime histogram.

CONCLUSION

In conclusion, the experimental library saves a considerable amount of time in this use case. The fact that the connections are not closed immediately reduces the overhead. The use of std::future with parallel communication corresponds to modern hardware utilization. The classic DOOCS library also offers asynchronous connections with "monitors". However, these connections are not intended for one-time communication. The experimental library offers a similar function. However, this functionality needs to be better implemented. Therefore, it has not been discussed in this paper. When planning the objects, it is a good idea to implement them independently as possible. This way, automatic tests can be used to ensure the quality of the software. The amount of work time required for such a project should not be underestimated. If it is not really necessary, it is always recommended to use already finished software.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP09

SMART VIDEO PLUG-IN SYSTEM FOR BEAMLINE OPERATION AT EMBL HAMBURG

E. Galikeev†, U. Ristau, C. Blanchet, D. v. Stetten, V. Palnati, S. Fiedler
all European Molecular Biology Laboratory (EMBL), Hamburg Unit
c/o DESY, Hamburg, Germany

Abstract

Fast data collection, image processing, and analysis of video signals are required by an increasing number of applications at the EMBL beamlines for structural biology at the PETRA III synchrotron in Hamburg, Germany.

Consequently, a new Smart Video Plug-in system has been designed in-house to meet the needs by combining video capture, machine learning, and computer vision with online feedback for motion control. The new system is fully integrated into TINE: data acquisition, and experiment control system.

In this paper, the architecture of the new video system is described and use cases relevant to beamline operations are presented.

INTRODUCTION

Recent projects carried out at EMBL Hamburg synchrotron beamlines for structural biology showed the necessity for a flexible tool that combines modern image processing libraries, video codecs, and motion control support for such experimental workflows like:

- Sample positioning using piezo stage control
- Control of the focus of the sample observation optics
- Image processing (e.g. filtering, denoising)
- Object detection and image recognition using machine learning techniques.
- Fast video recording and compression for offline and online sample motion analysis

The usage of the OpenCV library for image detection and computer vision at synchrotron beamlines was studied by Gofron and Watson [1] who showed that it can be applied for sample detection and centering robotic mounting, and evaluation of x-ray beam properties.

TINE at PETRA III [2] is a cross-platform, scalable distributed control system covering the majority of beamline operations and can be used with various OS like Windows, MAC/OS, and Linux. One of the core concepts of the TINE system is the "TINE Server" which represents a unique entity in the control network. It exports properties as endpoints used by the other TINE servers and clients to communicate with each other.

The new video system uses the full potential of the TINE API to provide means for the building and deployment of a new server as well as utilizing the centralized TINE alarming and archiving functionalities.

ARCHITECTURE

There are several reasons for choosing the Microsoft .NET framework [3] as a platform for integrating 3rd party functionality and user algorithms into the TINE system:

- .NET is a free and open-source, cross-platform software development framework.
- It enables interoperability between native C/C++ to preserve and take advantage of existing investments in unmanaged code.
- There exists wide official API support of the .NET platform among motion control and video solutions manufacturers.
- TINE .NET API allows getting full access to the EMBL beamline control system [4].
- C# is a powerful, modern, object-oriented type-safe programming language for rapid application development.
- Bindings for the NumPy and OpenCV libraries as well as such C# language features like LINQ provide extensive data processing toolsets for the experiment scripts.

The plug-in architecture [5], as shown in Fig.1, was chosen to meet the needs of the flexibility of adding new functionality (e.g. new camera or motion controller API support). The plug-in modules are meant to be separate containers for the additional features or user code that is designed to enhance the main functionality.

The core system provides interfaces to which plugins can connect, creating TINE server instances and routing data exchange between the plug-ins, and offering external communication with clients using standard TINE data transfer mechanisms. Also, the core system provides logging, native 3rd library support, and system configuration functionality.

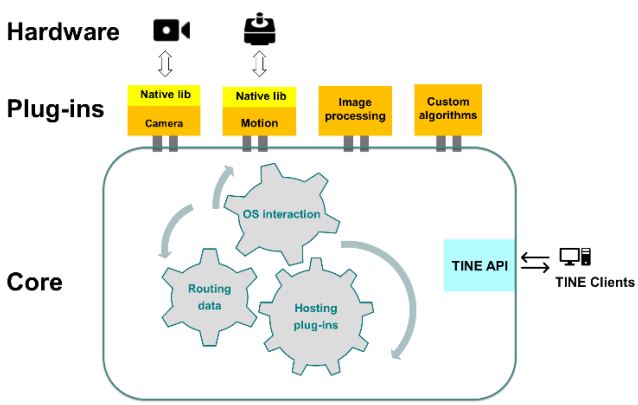


Figure 1: Architecture of the Smart Video System.

†emil.galikeev@embl-hamburg.de

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP09

Microsoft .NET framework supports the dependency injection (DI) software design pattern, which is used for achieving Inversion of Control (IoC) between classes and their dependencies. This pattern allows class A to call methods on an abstraction that class B implements, making it possible for class A to call class B at run time, but for class, B to depend on an interface controlled by class A at compile time (thus, inverting the typical compile-time dependency) [6]. At run time, the flow of program execution remains unchanged, but the introduction of interfaces means that different implementations of these interfaces can easily be plugged (e.g. new video cameras or motion controllers) within one application.

The decoupled architecture allows easy unit testing using mock objects for simulation of the behavior of complex real objects. It is extremely useful in the case of testing experimental scripts without involving the full hardware data acquisition loop.

Each plug-in implements a certain device or library-specific logic.

Video Camera Plug-In

- Receiving raw images captured from specific cameras
- Video streaming into the beamline control environment
- H.264 compression and transfer into the video storage for the offline sample motion analysis.

Image Processing Plug-in

- Processing raw images using OpenCV
- Receiving inputs for image processing algorithms from configuration and user interface
- Providing processed output for different consumers (e.g. lens autofocusing, sample positioning).

Motion Control Plug-In

- Piezo stage control using native libraries of SmarAct, PI, and other vendors
- Stepper and DC motor control via the Common Device Interface (CDI) in TINE.

USE CASES

Sample Exposure Unit at P12 Beamline for Small Angle X-ray Scattering (SAXS) Experiments

The design and installation of the new multi-purpose sample exposure unit (SEU2B) for scanning small-angle scattering experiments at the P12 SAXS beamline was carried out in collaboration with EMBL teams in Hamburg and Grenoble and with ESRF. The SEU was designed to accommodate different types of samples and sample holders such as bones and tissues for scanning SAXS experiments or microfluidic chips. One of the main purposes of SEU is to observe the microscopic sample while it is exposed to an X-ray beam during scanning. The installation (see Fig 2a,b) includes an on-axis sample observation camera (UEye, IDS GmbH, Germany) with a motorized lens system (Precise Eye Fixed Lens System, Navitar, USA) and a piezoelectric stage with motion control (Physik Instrumente, Germany). With the on-axis camera, the area of the sample to be scanned and exposed can be visualized during the SAXS experiments. Images of a microscopic sample can be collected in parallel with scanning SAXS measurements and microfluidic cells can be precisely aligned. Beamline control is available via the graphical user interface BECQUEREL [7], a Linux QT5-based application (see Fig 3). User experiment scripts are executed via the "meta server" which communicates with the Smart Video System via exported properties of the TINE Server.

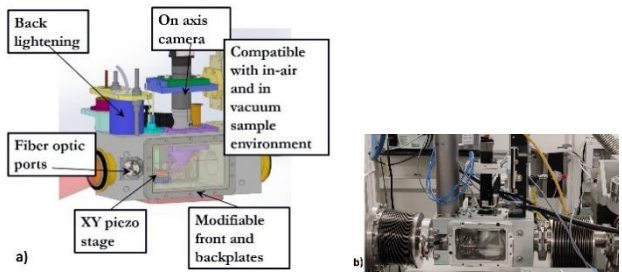


Figure 2: Sample Exposure Unit a) Model, b) Installation at the Beamline.

Video camera, image processing, and motion control plug-ins were added as part of the system configuration, providing manual and automatic focus control, and video streaming with image processing support for the beamline users. Currently, the piezoelectric stage control with 1D and 2D scanning is implemented to run directly through the BECQUEREL suite and the transfer of this functionality to the new system is underway.

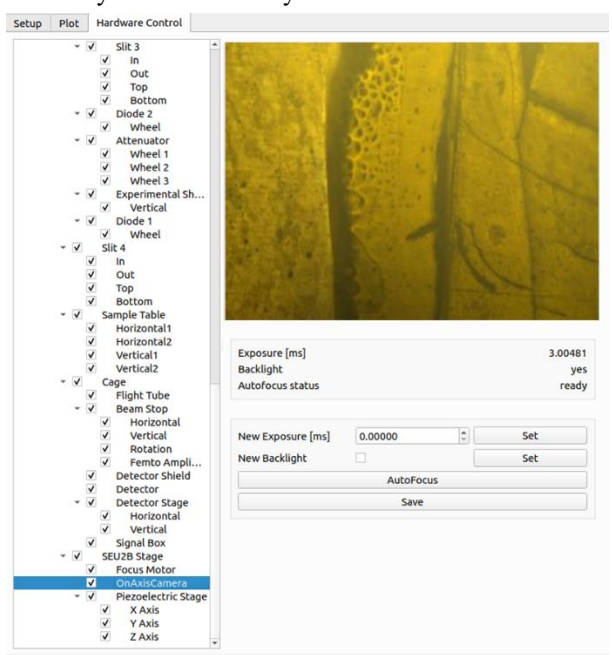


Figure 3: BECQUEREL GUI (SEU2B camera control tab).

Beamline P14.2 Time-Resolved Crystallography Experiments

Time-resolved experiments at the P14.2(T-REXX) beamline require sample delivery with fast-moving patterned silicon chips (~30 sample positions per second).

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP09

Fast video recording is essential for correct optical sample observation. For this purpose, a new high frame rate (~1000 fps) camera (XStream, IDT, USA) was installed and a beamline video server was developed using the TINE C++ framework. The server includes fast video recording and streaming capabilities with full control from the MXCuBE graphical user interface (see Fig 4). Most of the functionality of this new video server is currently being integrated into the camera plug-in. In addition, a motion control plugin for the SmarAct precision motion control system [8], to be used in the sample alignment part of the experiment scripts, is planned to be developed. Video compression is achieved via H.264 encoder from VideoLAN [9] with the video files automatically transported into the storage.

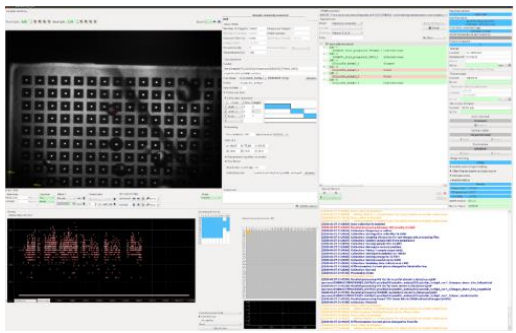


Figure 4: MXCuBE, Beamline Control Software for Time-Resolved Experiments at P14.2 Beamline [10].

SUMMARY AND FURTHER DEVELOPMENTS

The new TINE video system, developed with Microsoft .NET technologies, has shown to be a viable solution for current imaging and motion control tasks at EMBL Hamburg. Currently, the new system is under commissioning and intensive testing. Also, new features are being considered. Based on the feedback from beamline users, we plan to add new functionalities like the detection of sample distribution anomalies in T-REXX and the detection of misalignments of sample pins for the MARVIN robotic sample changer system [11]. Applying GPU solutions to accelerate the performance of resource-hungry image processing and machine learning tasks is being envisaged as well.

ACKNOWLEDGEMENT

The authors would like to thank the EMBL Grenoble instrumentation team for the collaboration in the SEU2B project and Andrey Gruzinov, DESY for his assistance in testing the new product.

REFERENCES

[1] K. Gofron and W. Watson, "Computer Vision at NSLS-II synchrotron beamlines", *Research Gate*, 2016. doi:10.13140/RG.2.2.24463.79524

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

[3] Microsoft .NET documentation, https://docs.micro-soft.com/

[4] U. Ristau *et al.* "The EMBL Beamline Control Framework BICFROK", in *Proc. PCaPAC'14,* Karlsruhe, Germany, WPO 012, pp.60-62.

[5] M. Richards *"Software Architecture Patterns"*, O'Reilly - Media, Inc., 2015.

[6] S. Smith "*Architecting Modern Web Applications with ASP.NET Core and Azure",* Microsoft Corporation, 2022.

[7] N.R. Hajizadeh *et al.*,"Integrated beamline control and data acquisition for small-angle X-ray scattering at the P12 BioSAXS beamline at PETRA III storage ring DESY", *J. SynchrotronRadiat.*, pp 906-914, 2018. doi:10.1107/S1600577518005398

[8] SmarAct website, https://www.smaract.com

[9] VideoLAN website, https://code.videolan.org/videolan/x264

[10] M. Oscarsson *et al.,* "MXCuBE2: the dawn of MXCuBE Collaboration", vol. 26, *J. Synchrotron Radiat.*, pp. 393-405. 2019. doi:10.1107/S1600577519001267

[11] U. Ristau *et al.,* "Marvin Update' the Robotic Sample Mounting System at the Embl-Hamburg", in *Proc. PCaPAC'18*, Hsinchu City, Taiwan, Oct. 2018, pp. 163-166. doi:10.18429/JACoW-PCaPAC2018-THP03

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP12

PLC OPERATED PLUG AND PLAY VACUUM GAUGE FUNCTIONALITY AT THE ARGONNE TANDEM LINEAR ACCELERATOR SYSTEM*

K. Bunnell†, C. Dickerson, B. Nardi, D. Novak, D. Stanton
Argonne National Labs, Chicago, IL, USA

Abstract

ATLAS (Argonne Tandem Linear Accelerating System) accelerator at Argonne National Laboratory is upgrading the vacuum control system from hardware-based, embedded controllers to modern flexible PLC-based controllers. This PLC (Programmable Logic Controller) system includes additional fail safes and a new remote operation feature. As part of this upgrade, enabling easy vacuum equipment replacement became apparent, specifically the vacuum gauges which are interfaced using serial communications. We developed a solution to remotely program the gauges, which offers more options for gauge control than the hardware-based controllers. Expanding on this, we added a process to initialize and restore the configuration for replaced gauges. This simplifies the process and eliminates the need for a system expert for these tasks.

INTRODUCTION

The ATLAS accelerator is located at the United States Department of Energy's Argonne National Laboratory in the suburbs of Chicago, Illinois. It is a National User Facility capable of delivering ions from hydrogen to uranium [1] for low energy nuclear physics research to perform analysis of the fundamental properties of the nucleus.

The accelerator requires pumps to create a vacuum in the beamline to prevent collision of the beam with air particles. Most of the vacuum control system uses custom built hardware-based chassis (Figure 1) that provide static vacuum interlock logic. The interlock logic largely relies on the set points of vacuum gauges to determine when safe operating conditions have been exceeded so that pressure-sensitive pumps can be isolated or stopped. The set point values can be locally adjusted from the gauge controllers. These systems are connected to a CAMAC (Computer Automated Measurement and Control) Serial Highway [2] to provide basic vacuum information to the control system such as: absolute vacuum pressures, pump status, and valve status.

Since 2019, ATLAS has been replacing the hardware-based vacuum monitoring system with flexible Schneider Electric PLC-based systems. The new systems enable remote vacuum control (Figure 2), flexible logic and vacuum hardware additions, and fully automated vacuum systems; all of which were cumbersome with the hardware-based system [3].

A goal of the ATLAS vacuum upgrade is to integrate vacuum automation. Since safety checkpoints are programmed in automated vacuum systems, there is less knowledge required to operate the system. At the same time, the new vacuum upgrade should have all the features of the hardware-based system. In this paper we will focus on seamless gauge replacement and set point configuration.

HARDWARE DESCRIPTION

The existing ATLAS vacuum system includes many obsolete vacuum gauges which feature manual set point adjustments from a gauge controller. The replacement devices largely include faster responding vacuum gauges with larger vacuum ranges, more features, and remote communication interfaces. To minimize the upgrade efforts and to keep all the former features in place, the obsolete vacuum gauges are being replaced by modern Granville-Phillips (GP) 275 and 390 gauges as the sectioned PLC upgrades take place. The modern gauges are modular and have integrated controllers. Therefore, when a gauge is replaced, all programmed information from the old gauge must be sent to the new gauge.

SOFTWARE DESCRIPTION

The PLC used for this vacuum upgrade features RS232 and RS485 serial communication natively. This is important since the majority of the new GP vacuum gauges require RS485 serial communication as their only interface to program the gauge, and read vacuum pressures. The default factory serial communication settings for all gauges are the same and will not be changed for use with the PLC.

Figure 1: Hardware-based vacuum Control Chassis with adjustable vacuum set point switches.

Gauge Addressing

Each GP vacuum gauge has a manual, 16-position adjustment of 0-F for device addressing. In addition, there is a programmable address offset that can be changed to a value of 0, 10, 20, or 30 (HEX). For PLCs with more than 16 gauges, the address offsets will be required to prevent duplicate addresses. With address offsets, a maximum of

* This work was supported by the U.S. Department of Energy, Office of Nuclear Physics, under Contract No. DE-AC02-06CH11357. This research used resources of ANL's ATLAS facility, which is a DOE Office of Science User Facility.
† kbunnell@anl.gov

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP12

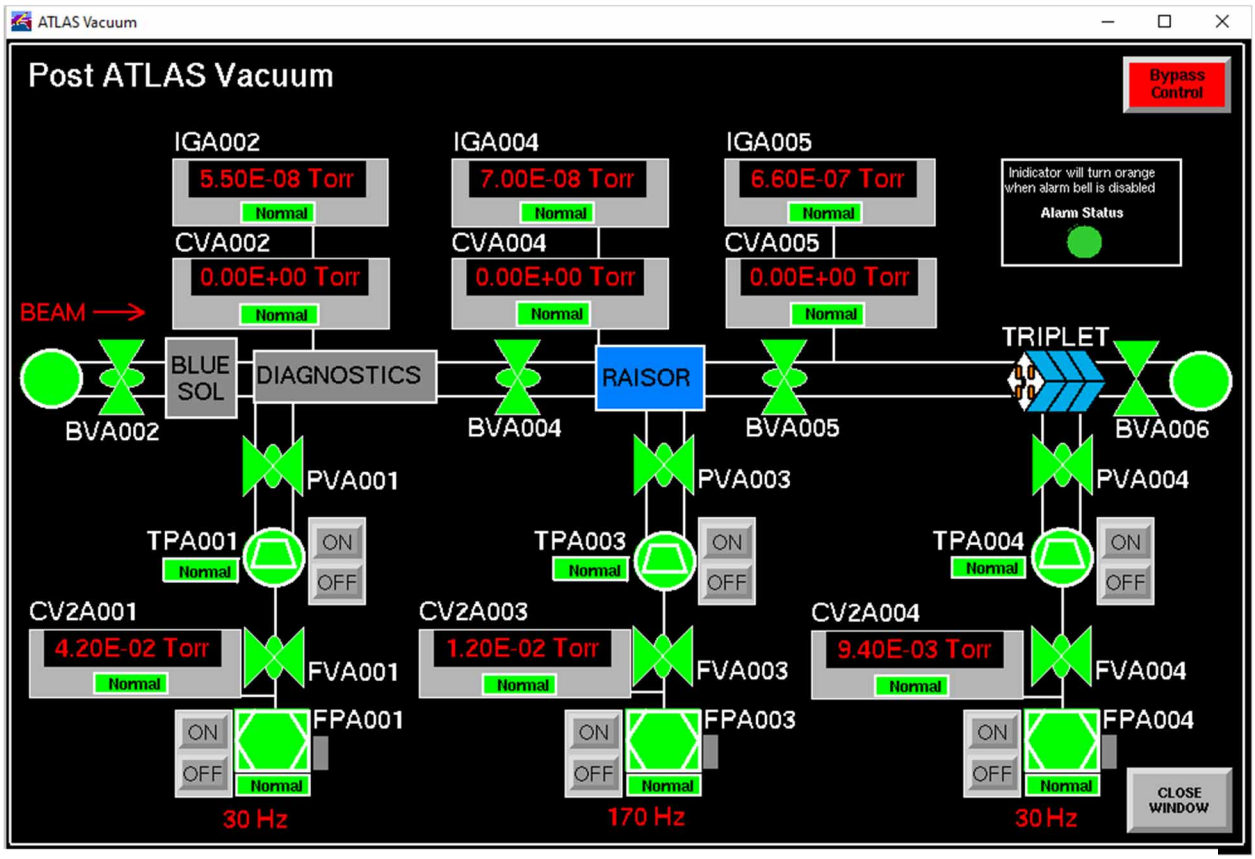


Figure 2: A typical ATLAS PLC-controlled vacuum display using Vista-Controls [4] VDraw feature.

64 devices are possible. However, addresses 0-F (where the address offset is zero) will be reserved for uninitialized devices. Meaning, that if a gauge needs to be replaced, the process will be to set the manual external switch on the new gauge to match, then let the PLC update the address offset over serial. This system, then limits our available addresses to 48 per PLC since the remaining 16 addresses are reserved for uninitialized devices. To change the address, some GP gauges require the toggle lock command to be sent. To get the address to take effect, GP requires that the gauge be powered off, or receives the reset command.

Required Serial Description

To duplicate the features of the hardware-based system on all GP gauges, the following command descriptions are needed: read vacuum pressure, reset module, toggle lock, read set point, and write set point. With the new system, technicians can update 1 set point at a time from a remote display interface. After every set point change, the appropriate set point value is updated in the PLC. After replacing a broken gauge, a gauge reinitialize button must be pressed to reset all the set points of a gauge. This differs slightly from the old system since the set points were originally set by front panel knobs on the gauge controller.

Figure 3 demonstrates the order of programming requests that the PLC will send to the vacuum gauge. The PLC operates using a state machine, which means that the serial communication will take several PLC cycles to complete, while the critically important interlock logic is processed during every PLC cycle. After the user requests a gauge to initialize using the display in Figure 4, the PLC first sends all the set points to the new gauge. At this point, the PLC sends a request to a gauge with the uninitialized address (the manual dial address only). If no gauges respond to this request, the PLC will then send a request to the initialized address (the manual dial address with the address offset). After several failed attempts at both addresses, the PLC will report an error.

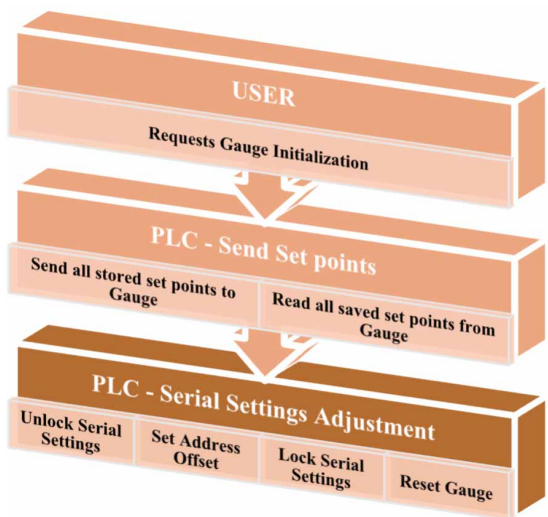


Figure 3: The order that a GP gauge is programmed by the PLC after a user requests an initialization.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP12

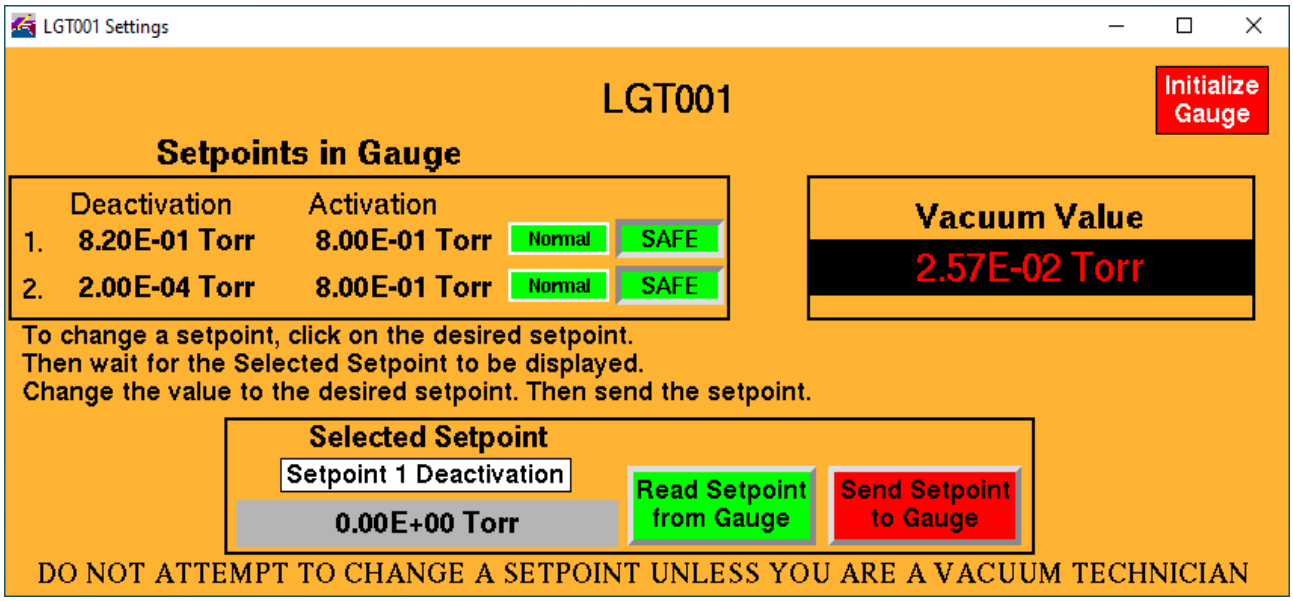


Figure 4: A Vista-Controls [4] concept display that features working set point adjustments and gauge initialization.

If the PLC finds a device with either address, it moves on to send the set point values to the gauge. Then, if the PLC is using the uninitialized address, the PLC will send the requests needed to add an address offset and force the gauge to apply the new address. All later requests to the initialized gauge will use the initialized address.

SETPOINT STORAGE

As mentioned, there is a copy of the set points that exist in the PLC and in each of the gauges. There is no way to save the set points in the PLC after a power bump except to purchase an expensive PLC-compatible SD card. The set points that are stored in the gauges, however, are non-volatile. Therefore, the PLC was designed to prioritize the set points in the gauges. At startup, the PLC will read every set point from each gauge to store them locally and display the value to technicians. If a gauge needs replaced, and an initialization is performed on the gauge, the saved set points in the PLC are sent to the gauge. With this system, there is a low chance the PLC power cycles before a gauge can be replaced; In which case the set points for the gauge will be manually entered again.

USER EXPERIENCE

As mentioned previously, there are plenty of advantages of PLC-based vacuum control over the existing hardware-based method. One important advantage is that the PLC-based method allows for remote control and monitoring of all vacuum devices. Each of these control displays can be accessed from any control system display computer at ATLAS. For select individuals, the control displays can also be accessed from outside the lab.

FUTURE PLANS

Automated gauge configuration is currently in the testing phase and will be installed with a new vacuum upgrade section during this winter's accelerator shutdown. After this install, it would be ideal to create a system that prevents the vacuum set points from being lost, should this become an inconvenience to the operations staff.

With the new setup, each GP gauge is required to be factory reset by a control system expert when there is a chance that it re-enters service. New gauges are already at a factory reset state, but if gauges are removed from the beamline, they will still be programmed to their previous service settings. Before the gauge enters service again, the address offset should be reset to allow the initialization algorithm to function properly. To match the theme of building systems that allow all of this to be done by technicians, a device could be built that will send the factory reset command to all possible serial addresses.

CONCLUSION

The purpose of this work is to provide vacuum technicians the same functionality as the previous hardware-based vacuum control system. The new system will also offer the advantage of full gauge replacements and remote control of vacuum gauges that was otherwise impossible. Though the newly upgraded system is not installed yet, small scale test of this system have proved effective. Ultimately, this upgrade will provide ATLAS with a complete feature replacement of the previous hardware-based system as well as lay the groundwork for future state-of-the-art vacuum automation.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP12

ACKNOWLEDGEMENTS

We would like to thank Arnold Germain for sharing his knowledge of vacuum systems, assisting us with the vacuum test setup, and assisting us to setup goals for this vacuum upgrade.

This work was supported by the U.S. Department of Energy, Office of Nuclear Physics, under Contract No. DEAC02-06CH11357. The research used resources of ANL's ATLAS Facility, a DOE Office of Science User Facility.

REFERENCES

[1] "ARGONNE TANDEM LINAC ACCELERATOR SYSTEM Available Beams", https://www.anl.gov/atlas/available-beams (visited 2022-09-20)

[2] IEEE Std. 595-1982, Standard Serial Highway Interface System, The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017.

[3] C. Peters *et al.,* "PLC Based Vacuum Controller Upgrade and Integration at the Argonne Tandem Linear Accelerator System", in *Proc. ICALEPCS'17*, Barcelona, Spain, October 2011, pp. 724-727. doi:10.18429/JACoW-ICALEPCS2017-TUPHA131

[4] Vista Control Systems, Inc. https://www.vista-control.com/

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP13

CONTROL AND TIMING SYSTEM OF A SYNCHROTRON X-RAY CHOPPER FOR TIME RESOLVED EXPERIMENTS

U. Ristau, V. Palnati, J. Meyer, D. Jahn, S. Fiedler
European Molecular Biology Laboratory, EMBL, Hamburg, Germany

Abstract

A mechanical X-ray chopper for the Small Angle Scattering (SAXS) beamline P12 operated by the EMBL Hamburg Unit at the PETRA III synchrotron on the DESY campus has been developed. In this paper we will describe how control and timing for time resolved experiments have been implemented.

INTRODUCTION

In the time resolved SAXS experiments carried out at P12, the structural changes of proteins in solution are studied. The scattering of the sample is recorded with a fast area detector as a function of time after a reaction has been triggered. As the samples are sensitive to radiation damage, it is desirable to minimize the exposure to the X-ray beam between two frames. This is achieved with the chopper that can be adjusted to variable duty cycles.

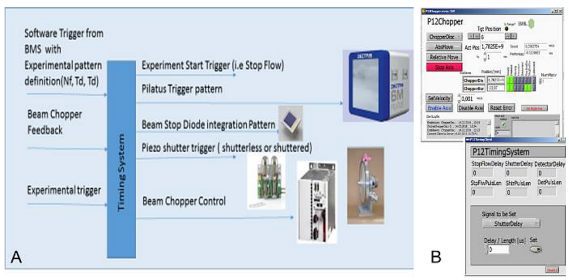


Figure 1: (A) Schematics of instruments to be synchronized for an experiment. (B) Clients for the timing control of the experiment and of the chopper [1].

Also pump and probe measurements can be carried out with chopper systems where the sample is probed with a short X-ray pulse after a laser excitation [2]. For such experiments several systems or processes must be synchronized like storage ring bunch pattern, sample injection, excitation laser pulse, intensity monitor signal, shutter opening, detector trigger, and the chopper revolution frequency that must be kept very constant (Fig. 1A). The solution chosen for the timing and synchronization system is based on EtherCAT electronics [3]. Servers and clients are integrated into the TINE control system [4] with LabView [5] (Fig. 1B) and TwinCAT [6].

CHOPPER DESIGN & IMPLEMENTATION

Different designs for X-ray beam choppers used at synchrotrons have been suggested ranging from the milli- to the sub-microsecond range (see for example [2, 7]).

Our beam chopper design (Figs. 2A, 2B) is based on a rotating disk that has a 10-fold symmetry cut-out pattern with stepped slots of different length in angular direction (Fig. 2C). When the disk is rotating the beam is chopped as a function of

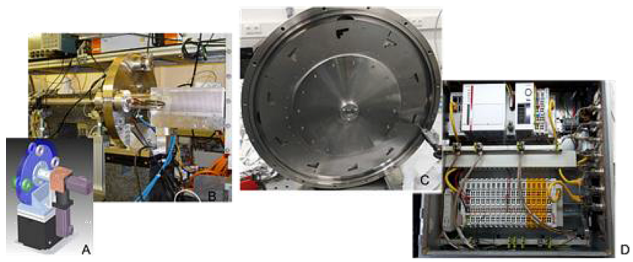


Figure 2: (A) Mechanical design of the chopper, (B) Photograph of installation at P12 beamline, (C) Photograph of the chopper disk, (D) Fieldbus electronics for chopper control.

the length of the slots. The chopper can be translated laterally so that the X-ray beam can be transmitted at positions with different slot lengths. In this manner variable duty cycles can be adjusted. The chopper can also generate short X-ray pulses. For this, a narrow slot at the outer diameter of the disk is used. The pulse length can be varied by adjusting the angular velocity.

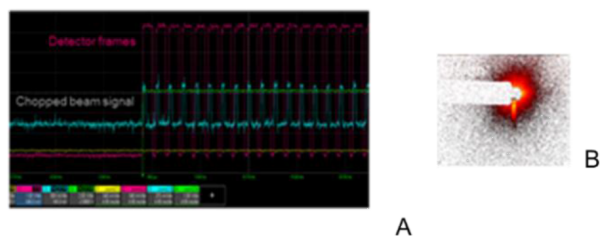


Figure 3: (A) Chopped beam signal at 750 fps measured with an intensity monitor at the detector with ¼ duty cycle. (B) Residual parasitic scattering pattern in the SAXS region of the chopped beam at 750 Hz.

Examples of the system in operation are depicted in Fig. 3. The chopper can run with a beam chopping frequency of up to 1500 Hz. A frequency stability of $<10^{-5}$ can be achieved and the jitter is below 2 µs. The shortest X-ray pulse that can be produced is 10 µs.

CHOPPER CONTROLS

The TINE Control system is used as transport layer at the EMBL Hamburg beamlines at PETRA III [1, 8–10]. All server and client communication are based on the TINE protocol. The TINE CDI (Common Device Interface) [11] has

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP13

a plugin for ADS (Automation Device Specification) communication with TwinCAT. ADS is the TwinCAT internal communication protocol to link IOs, NC (numeric control) and the PLC tasks. TwinCAT and TINE servers are both running embedded on a DIN-Rail PC. The communication between the real time PLC and the control system is executed as fast local loopback data traffic. The TwinCAT real time software PLC is executed on the real time kernel of the Intel PC. The TwinCAT real time Operating System (OS) and the Window OS share the PC hardware resources. 80% of the CPU resources for the PLC task execution are attributed to the standard TwinCAT installation used by EMBL. The TwinCAT development environment and the TINE installation are installed on the Windows partition. They have access to 20% CPU resources in this configuration. The PLC and NC host the critical processes for real time operation and communication including the IOC data traffic based on the EtherCAT [3] protocol.

After programming the TwinCAT PLC, code is downloaded to the real time OS part of the PC and executed in the real time partition. The real time EtherCAT protocol passes through all field bus modules and exchanges data with the IOC modules on the fly.

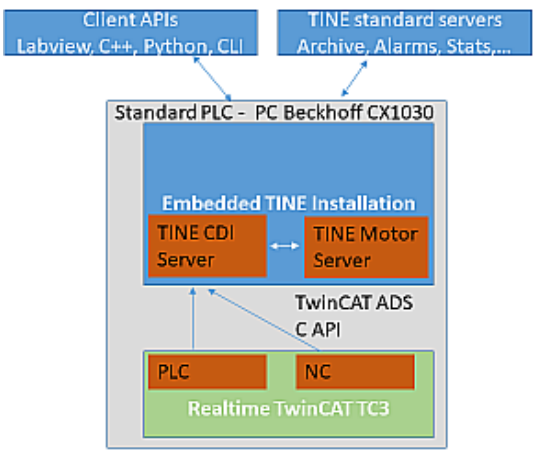


Figure 4: Motion control architecture.

ADS is also the API for communication between Beckhoff [4] and the TINE protocol. The bus plug for TwinCAT based on ADS communication gives access to all defined ADS variables, if configured. Fast access to IOC data is integrated through this interface. Motion control under TINE is performed with the TINE Motor server. The TINE Motor server is equipped with a TwinCAT NC interface (see Fig. 4).

CHOPPER MOTOR AND MOTOR CONTROLLER

A fast rotation 4-pair DC motor with 9000 RPM has been chosen for driving the chopper wheel (AM832, Beckhoff). The corresponding motor controller (AX5000, Beckhoff) operated by the TwinCAT NC offers different motion profiles. The "velocity control" profile has been chosen and implemented for the chopper application.

The rotation velocity of the motor runs in a feedback loop where the drifts of the velocity are corrected with respect to the nominal frequency supplied by an external frequency generator. For this purpose, a function generator (HP3314) has been implemented. Alternatively, the bunch clock of the PETRA III synchrotron, provided by DESY, has been integrated in a second step of the development and can be used for the velocity control and the timing of the experiment. Due to the insertion of the bunch clock, a synchronization of the data acquisition with respect to the revolution frequency of the PETRA III electron beam can be achieved (see schematic in Fig. 5).

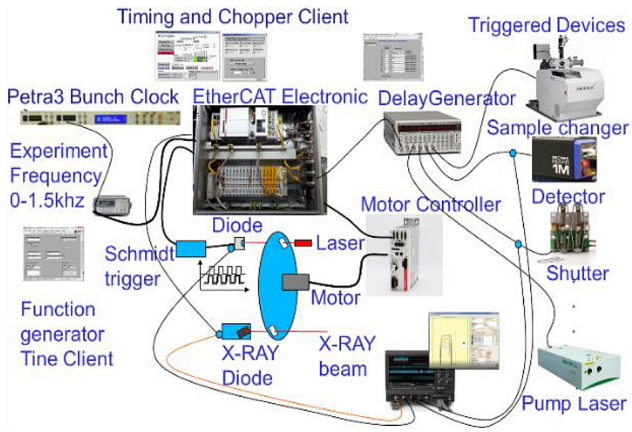


Figure 5: Layout of chopper control including connectivity for timing of the experiment.

A diagnostic laser has been installed at the chopper disk in order to monitor the speed of the disk. The laser beam is chopped by the same slot pattern of the disk as the X-ray beam. This way, the signal of the X-ray and the laser beam have the same timing pattern. The signal of the chopped laser beam is detected by a fast photo diode (Centronic) that has a signal rise time of <2 ns/100 mV and an afterglow of less than 10 µs. The diode signal is then fed into an adjustable TTL Schmitt trigger. The TTL output signal of the Schmitt trigger indicates when the X-ray beam is blocked or transmitted by the chopper. The advantage of using this additional diode signal is its fast readout and to have the information for synchronization available also without the X-ray beam permanently available. The feedback control is implemented in the TwinCAT PLC for which the fasted available cycle time has been adjusted to 50 µs whereas the NC cycle time is 2 ms. With these parameters, drifts of the disk velocity can be corrected.

TIMING SYSTEM

The timing system, implemented as PLC in TwinCAT, is based on an XFC (extreme fast control) module of Beckhoff. The module can generate outputs with nanosecond accuracy. The distributed clock functionality of the Beckhoff system has been used to generate precise trigger timings for the connected devices like detector, laser, or sample delivery system. Depending on the experiment, the timing system

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP13

offers different modes of operation for which different setups are used. The trigger distribution is based on a frequency divided input signal generated by the bunch clock.

Bunch synchronous outputs with 100 Hz, 1 kHz, and 10 kHz frequencies are supplied by the frequency dividers. These signals are used to create the master trigger signals for the experiments which in turn serve as input signals for either an EtherCAT delay generator (EL1258/EL1259) or, as a high-performance alternative, a pulse and delay generator (StanfordResearchSystems DG 645) with up to 4 outputs with picosecond accuracy.

In "variable frequency" control mode the frequency generator creates master trigger pulse patterns which can be delayed by the trigger distribution. A single trigger or series of triggers can be generated.

In "free running" mode the frequency is defined by the EtherCAT master clock. A start pulse to trigger the experiment is provided as input to the trigger distribution module.

CONCLUSION & OUTLOOK

The chopper system has been implemented into the setup of the P12 beamline and is available for time resolved SAXS experiments for the user community. The different slot lengths of the chopper disk in combination with the possibility to vary the speed of the chopper motor enables a flexible timing for each individual experiment. The timing system provides trigger outputs for several devices in different experimental modes with variable pulse length and the option to run with variable repetition rates.

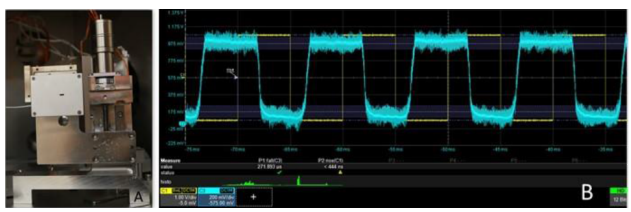


Figure 6: EMBL/SmarAct fast shutter operated at maximal repetition rate of 100Hz at the EMBL P12 Beamline.

The functionality of the system can be further enhanced by inserting an improved fast beam shutter. By combining the X-ray chopper with a millisecond shutter it will be possible to isolate single exposures or pulse trains determined by the exposure length of the chosen chopper slot. A prototype of such a shutter (Fig. 6) has been designed and built in collaboration with the company SmarAct [12] specialized on piezoelectric applications. The shutter has an opening/closing time of 500 µs.

REFERENCES

[1] U. Ristau *et al.*, "The EMBL Beamline Control Framework BICFROCK", in *Proc. PCaPAC'14*, Karlsruhe, Germany, Oct. 2014, paper WPO012, pp. 60-62.

[2] T. Graber *et al.*, "BioCARS: a synchrotron resource for time-resolved X-ray science", *J. Synchrotron Radiat.*, vol. 18, pp. 658-670, 2011. doi:10.1107/S0909049511009423

[3] EtherCAT, https://www.ethercat.org

[4] TINE (Three-fold Integrated Networking Environment), https://tine.desy.de

[5] National Instruments, https://www.ni.com

[6] Beckhoff, https://www.Beckhoff.com

[7] M. Gembicky *et al.*, "A fast mechanical shutter for submicrosecond time-resolved synchrotron experiments", *J. Synchrotron Radiat.*, vol. 12, pp. 665–669, 2005. doi:10.1107/S090904950501770X

[8] U. Ristau *et al.*, "The Concept of EMBL Beamline Control at PETRA III", in *Proc. PCaPAC'08*, Ljubljana, Slovenia, Oct. 2008, paper MOZ02, pp. 22-24

[9] R. Bacher, "The new Control System for the Future Low-Emittance Light Source PETRA3 at DESY: Sprinting to the Finish", in *Proc. ICALEPS'07*, Oct. 2007, Knoxville, TN, USA, paper TPPB27, pp. 217-219.

[10] U. Ristau *et al.*, "Control of the New EMBL-Hamburg Sample Changer", in *Proc. PCaPAC'08*, Ljubljana, Slovenia, 2008, paper WEP019, pp. 210-211.

[11] P. Duval and H. Wu, "Using the Common Device Interface in TINE", in *Proc. PCaPAC'06*, Newport News, VA, USA, pp. 17-19.

[12] Smaract, https://www.smaract.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP15

NEXT GENERATION GSI/FAIR SCALABLE CONTROL UNIT: LESSONS LEARNED FROM 10 YEARS IN THE FIELD

K. Lüghausen, M. Dziewiecki, K. Kaiser, G. May, S. Rauch, M. Thieme, GSI, Darmstadt, Germany

Abstract

The end-of-life of many components brought the need for a redesign of the main Control System Front-End for GSI accelerators - the SCU (Scaleable Control Unit). It was a chance to make improvements and use more powerful state-of-the-art core components. This included a new Arria 10 FPGA and a completely redesigned housekeeping circuit based on an AVR micro controller. Further, the project was cleaned by removing unused components and features. Main frame conditions stay fixed for backward compatibility, like the mechanical form factor or the 16-bit parallel bus. Majority of gateware and firmware could be reused and just some adaptations for the new FPGA were needed. Nevertheless, providing continuous compatibility with legacy peripherals needed a substantial effort.

INTRODUCTION

Scaleable control units (SCUs) have been designed as a uniform, intelligent, realtime interface between various electronic components of accelerators and the control network. Internally, they contain an off-the-shelf computing module (main CPU, main memory) running Linux and a custom baseboard (I/Os, White-Rabbit timing receiver) with an Intel Arria FPGA as summarized in Table 1. The integrated timing receiver allows executing various tasks with nanosecond-level precision while the main CPU covers higher-level computing and data management. On the network side, they offer a Gigabit Ethernet interface for communications and a White-Rabbit slave for timing and triggering. On the device side, they are equipped with a parallel bus and some further interfaces as shown in Fig. 1. The device is disc-less: all data, software and the boot image are delivered via the network.

The new generation, the SCU 4, keeps compatibility with previous versions while providing significantly more resources.

SCU4 HARDWARE DESIGN

Table 1: FPGA Comparison

Parameter	SCU3	SCU4
FPGA Type	Arria II	Arria 10
Technology	40 nm	20 nm
Logic Elements	100 k	270 k
BlockRAM	8121 Kb	17 452 Kb

The general design was inherited after the previous SCU generation [1]. The basic parameters like mechanical dimensions and electrical connection to the back-plane remained unchanged.

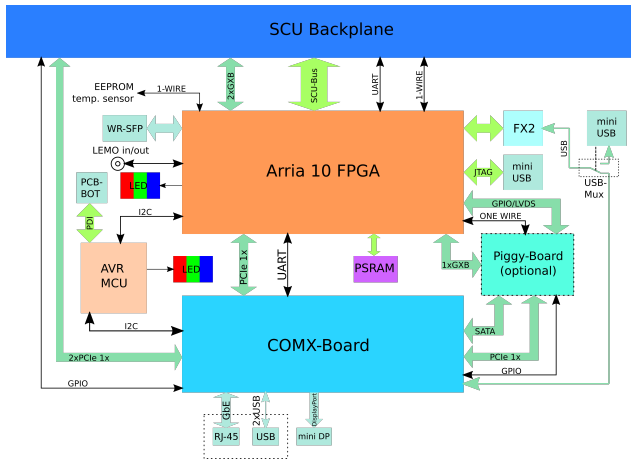


Figure 1: Block diagram of SCU4.

Major changes took place regarding the front panel. An OLED-display was removed. It turned out to be of limited use while it needed significant effort in assembly and occupied valuable space. Instead, a DisplayPort socket was added which allows connecting an external screen and a RGB-LED indicates the system status.

An important feedback from field operation was the lack of User-I/Os. As a result we added four additional user I/O lines. A dual mini D-Sub 9 debug connector was replaced by a mini USB device connector. A summary of these changes is shown in Table 2.

Table 2: SCU IOs

IOs	SCU3	SCU4
SCU-Bus	1	1
SFP	2	1
GbE Ethernet	1	1
USB 2.0 Host	2	2
USB 2.0 Device	–	1
Serial	2	–
Video	–	miniDP
LEMO 5V-TTL	2 In/Out	2 In, 4 Out
Logic-Analyser	1	–

The form-factor of the computing module has been changed from COMX Type 2 to Type 10 [2]. With a smaller footprint, it allowed us to rotate the module by 90 degrees, giving better airflow to cool the FPGA. The new module brings also advantages in processing power, as shown in Table 3.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP15

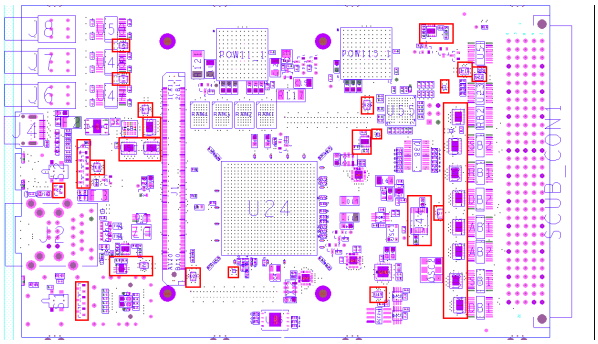


Figure 2: Top Layer: Red marked areas need for 1.8 V ↔ 3.3 V levelshifting (including device, keep-up and vias).

The communication between the main CPU and the FPGA is still done via PCIe. However, an additional connection via USB is provided, which offers shorter latency in case of single Byte/Word access operations.

Further, the external FPGA memory was updated. A CFI-Flash memory was removed as it was never used with the SCU3. The external FPGA DDR memory was replaced by a pseudo-SRAM. This has advantages for the FPGA pinning and synthesis and allowed reducing the number of power supply voltages.

Table 3: Computing Module

Feature	SCU3	SCU4
Form Factor	COM Express Type 2	COM Express Type 10
CPU	Intel N2600	Intel E3940
Cores	2 (4 HT)	4 (native)
CPU speed	1.6 GHz	1.8 GHz
RAM size	4 GB	8 GB
Geekbench 5.0		
Single-Core Score	89	285
Multi-Core Score	238	1065

One big difference between the Arria II and that Arria 10 is the Arria 10 cannot use 3.3V as I/O-Voltage anymore. On the other hand, a lot of legacy features need 3.3V voltage levels. This makes it necessary to add 1.8V↔3.3V levelshifters. In our case this requires nearly 7 percent of the PCB space as shown in Fig. 2.

VOLTAGE LEVELS AND POWER

The SCU has rather complicated power supply scheme, as shown in Fig. 3. To simplify the power supply, we decided to use a FPGA speed grade with the same voltage level for the core and the transceiver and just one voltage level for I/O. Using only 1.8 V as I/O voltage prevents the use of modern DDR-RAM as external memory.

The Arria 10 needs a special power sequence. This is provided by an ATxmega MCU-based circuitry. The MCU is also used for additional housekeeping tasks. In the early

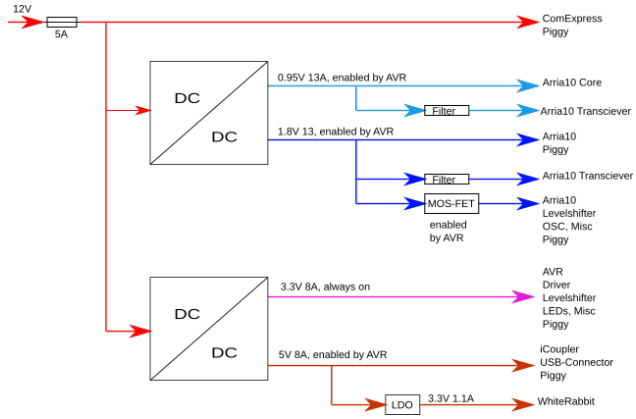


Figure 3: SCU4 power rails.

stage, a MAX 10 FPGA was planned for this job. Unfortunately, supply-chain issues brought the need to switch to an easier available chip.

PARALLEL AND SERIAL BUSSES

The main purpose of the SCU is the provision of the SCU bus. It is a 16-bit parallel backplane bus with 12 slots for slave cards. These slave cards come in a variety of types like ADC/DAC, GPIO, MIL-STD1553 master, controllers for magnet power supplies or controllers for RF systems. The interface definition of the bus was fixed before the SCU design started. This enabled other departments to develop slave cards for this interface.

The decision to use a a 16-bit parallel bus was based mainly on lots of experience with other parallel backplanes (8/16-bit), albeit with lower speeds. Another important design criteria was the possibility to analyze the bus transactions with an external logic analyzer. These were still widely in use when the design process started. In-System debug techniques like SignalTap from Altera or ChipScope from Xilinx where not yet used. This changed with the first prototypes of the SCU and In-System Debugging became a standard. The commissioning of the parallel backplane turned out to be quite difficult because of electrical interference. It took several versions of the backplane PCB and the addition of active termination to get the parallel bus into working condition. Even then, a few bus signals needed to be deglitched inside of the FPGA, first of all the dedicated interrupt signals.

The serial PCIe connection between the FPGA and the COM Express module was by far easier to build than the parallel backplane PCB. During the prototyping phase we used a modified PCIe card, where the PCIe lanes were rerouted with magnet wire. This enabled us to test the PCIe controller in the Altera FPGA and our PCIe connection worked perfectly in the first pcb version of the SCU. The PCIe controller is electrically based on an Arria transceiver. We used the same transceivers for a GbE connection (synchronous ethernet for timing) with an SFP. This was just as easy to build as the PCIe link.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP15

The SCU4 will have for the first time a USB connection to the FPGA which provides a UART interface for debugging and an Etherbone connection to the SoC inside the FPGA. It will replace two dedicated UART connectors. The USB interface is already used in timing receivers which use similar architecture and has proven to be reliable. The interface features USB2 and is based on the EZ USB FX2LP from Cypress.

SECOND USE CASE FOR SCU

It is planned to give the SCU hardware another use case as a timing message generator, called Data Master, for the General Machine Timing (GMT) System. Currently another hardware, based on a PCIe card, is used for this purpose. The SCU hardware has to be designed with this second use case in mind. This does mainly imply the possibility to be able to use a bigger FPGA with more internal memory. The difference will be roughly four times the size of the internal memory of the PCIe card. This will enable the Data Master to handle more and bigger schedules.

CONCLUSION

The maintainability is one of the major issues with electronic systems in 'big science'. With high level of complication and limited manpower, the process of designing and commissioning takes years. Thus, the lifetime should reach decades. On the other hand, continuously altering market of electronic components makes it difficult to produce devices only few years after designing. Keeping projects up to date while maintaining backward compatibility is getting a major challenge.

With its fourth generation, the SCU project can be considered mature. It's proven with already over 500 of running devices and it's free of major problems. On the other hand, using state-of-the-art components in SCU 4 extends the project lifetime by years. This gives a good perspective for retrofitting the old GSI's Unilac control system as well as providing control systems for the newly built facilities of FAIR.

REFERENCES

[1] M. Thieme, W. Panschow, S. Rauch, and M. Zweig, "Single Board Computer for Equipment Control in the FAIR Accelerator Control System", presented at ICALEPCS'09, Kobe, Japan, Oct. 2009, unpublished.

[2] "PICMG COM Express®Carrier Design Guide", PICMG, Rev.2.0, 2013. https://www.picmg.org/resources/design-guides/

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP16

OCELOT INTEGRATION INTO KARA'S CONTROL SYSTEM

P. Schreiber*, E. Blomley, J. Gethmann, W. Mexner, M. Schuh, A.-S. Müller
Karlsruhe Institute of Technology, Karlsruhe, Germany

Abstract

The Karlsruhe Research Accelerator (KARA) at the Karlsruhe Institute of Technology (KIT) is an electron storage ring and synchrotron radiation facility. The operation at KARA can be very flexible in terms of beam energy, optics, intensity, filling structure, and operation duration. For different aspects of the operation of the accelerator separate and individual simulation models are in place using different simulation tools, custom lattice data and varying levels of maintenance. In a general push at the accelerator to provide unified access via Python, a new framework was implemented using Ocelot with a much closer integration to the accelerator control system and supplementary tools. This allows a better integration and lowers the effort necessary for simulations and predictions of actual changes to the beam properties based on live data. It also provides a good entry point for the various Python based machine learning activities at the accelerator and the goal to obtain an easier to maintain and test accelerator model. This paper presents the taken approach and current status of this project.

INTRODUCTION

The storage ring KARA (Karlsruhe Research Accelerator) is operated as an accelerator test facility and serves as the KIT light source. It is a ramping electron synchrotron from 0.5 GeV to 2.5 GeV with a 4 fold symmetry and a total of 8 double bend achromatic cells. The circumference of 110.4 m and frequency of the RF system of 500 MHz results in a bunch revolution time of 368 ns and a bunch spacing of 2 ns. Furthermore, a wide range of beam diagnostics tools is available. A lot of beam time is dedicated to machine physics including experiments performed by scientists from external institutions. KARA is available as R&D facility for TNA (TransNational Access) via Eurolabs [1]. In order to make best use of the beam time simulations beforehand are necessary. Therefore easy access and ideally pre-set-up simulations are required.

REQUIREMENTS

Performing simulations for an accelerator, especially for new students or external scientists working at KARA can be hard to set up as many details such as exact lattice parameters as well as machine specific conversion coefficients are mandatory. Therefore the integrated simulation should be easy to use and have reasonable defaults. It should furthermore allow a reasonable amount of customisations to adapt the simulations to the various needs.

Furthermore, future systems might rely on e.g. the optics functions for the currently used settings at KARA. Such systems could consist of machine learning applications for various tasks such as operation optimisation. For these systems the simulation results should be readily available via network connections. The simulations for such systems should also be run periodically to ensure recent values.

A third requirement for the integration is easy maintain- and extendability. Additional simulation features, such as tracking or new elements should be easily implemented in order to encourage extension of the system.

SIMULATION SETUP

The simulation setup consists of a lattice defined with LatticeJSON [2], an epics-IOC [3] with records for each element or family, ocelot [4] as simulation software and a control panel in CSS [5]. An overview of the involved objects and their connections is shown in Fig. 1.

From the LatticeJSON definition an intermediate object is created that acts as uniform interface between the lattice definition, the IOC and the actual magnetic lattice for ocelot. This approach was chosen in order to allow potential other simulation tools to make use of the same lattice input. For each element in the lattice, the intermediate object creates an element type specific object. These element objects are responsible for transforming the user facing values of current flowing through the magnets to magnet strengths and also for creating the elements used by ocelot.

The intermediate lattice object is used by the automatic creation of an epics-IOC. A record in the IOC is created for each specified PV of each element in the LatticeJSON definition (this allows for multiple PVs per element if necessary).

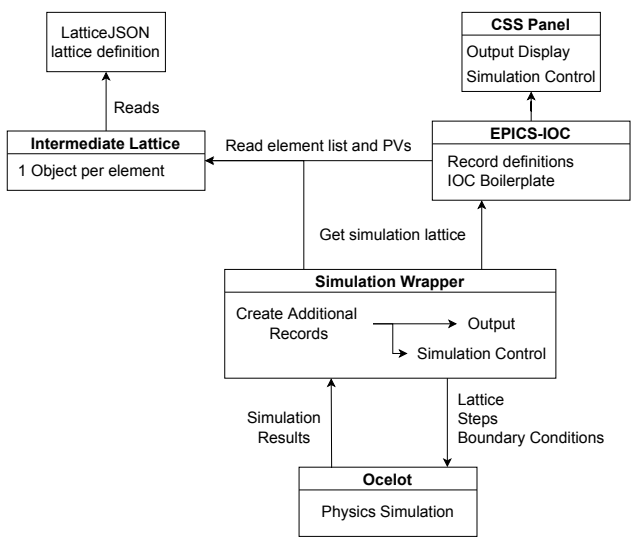


Figure 1: Structure and connections between different objects involved in the simulation integration system.

* patrick.schreiber@kit.edu

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP16

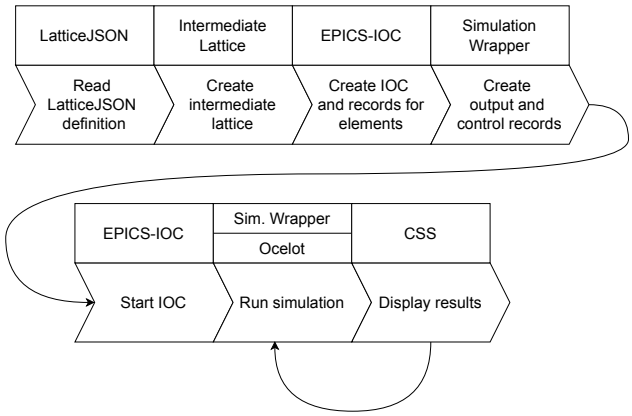


Figure 2: Control flow of the startup and simulation process of the simulation integration system.

These allow control over properties of the elements such as the magnet strength. However, the original LatticeJSON schema does not allow to specify PV names and lacks support for certain lattice elements such as BPMs or corrector magnets. As both the additional elements and the attributes for PVs are required, the LatticeJSON schema was extended to include them.

A wrapper around the simulation is used to manage all the previously mentioned elements. It creates additional records in the IOC to allow user control over the simulation such as the granularity of evaluation of the optics functions or the interval between automatic simulations. This wrapper also takes care of managing the simulation and updating the magnetic lattice for ocelot. The start-up order and flow is visualised in Fig. 2.

To conform to the general theme of using python for control system integration all elements are written in python, where for the IOC the pythonSoftIOC [6] package is used. This furthermore focuses on maintainability. As python is easy to read and is widely known it is relatively simple to add additional simulation features or fix bugs in the years to come.

To fulfill the requirement to have updated simulation values (e.g. optics functions or tunes) available for settings used at the accelerator at the moment (e.g. currents flowing through the magnets), periodic online simulations with updated values are run. These periodic runs are managed by the simulation wrapper with a customisable periodicity. For performance optimisations these simulations are performed with a fixed frequency but only when current input values are changed.

INTERFACE

The interface to the simulation is entirely handled via epics. PVs for elements with modifiable attributes allows control over the lattice. This includes the strength of quadrupoles, sextupoles and bending magnets. Additional PVs are provided to store the results or control the simulations. This includes calculated tunes and optics functions.

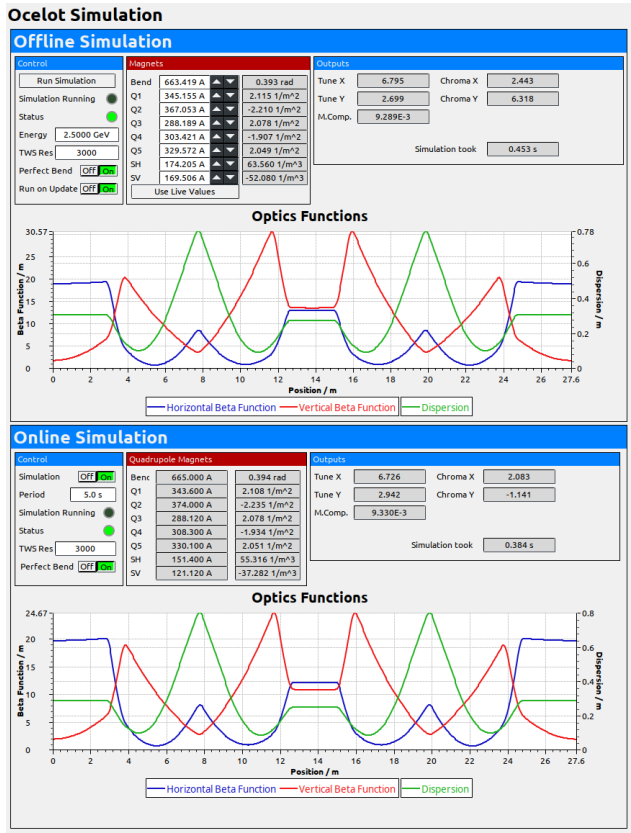


Figure 3: Screenshot of the user interface to the ocelot simulation. The shown panel is provided by CSS, the standard control system UI at KARA. Controls of the simulation as well as control over the magnets for the simulation is present, combined with the output values and optics functions.

Presently the control over the lattice is entirely performed via engineering units. This means the values for magnets are set as current in Ampere. This is also used for the simulations at the moment. It is possible to control the simulation via physical units as well, but the interface (extra PVs) is not available in the UI at the moment. The rationale behind this is to use the same interface for the machine as for the simulations to simplify the switch from simulations to the actual machine or vice versa. The magnet strengths in physical units is displayed in the UI, however, to help users accustomed to them.

To allow access to the simulations a graphical user interface was implemented based on CSS BOY. A screenshot of the interface is shown in Fig. 3.

The UI is split in two panels, the top panel shows the controls and results for offline simulation, where the user can manually select values for e.g. the magnet currents. The lower panel shows the controls and results for the periodic online simulations with limited control over the simulation as most values are taken from the currently used settings at KARA.

Some simulation results are displayed as numerical values, these include the tunes, chromaticities and the momentum compaction factor. The optics functions are displayed as a

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP16

combined plot for the beta function in both planes and the dispersion in the horizontal plane.

Apart from the obvious values for the magnets there are additional settings available such as the number of points used for evaluation of the optics functions as well as the beam energy. The latter is necessary to correctly convert from current flowing through the magnets to the magnet strength k. The UI offers to use the current of the dipoles or use a "perfect" magnet with a deflection angle equal to 360° divided by the number of bending magnets present in the lattice (22.5° at KARA). Furthermore it is possible to automatically run the simulation once the value for a magnet is changed to keep the simulation synchronised to the displayed values.

APPLICATIONS

Recent studies and efforts at KIT are aimed towards the usage of machine learning for autonomous accelerators. As the online simulation results are readily available via epics also within python, this system offers easy integration of the simulation results into such algorithms. In turn this allows the integration of machine learning applications into the control system. The centralised simulation would then reduce the computing power necessary as each application can rely on the same results and does not have to simulate individually. Furthermore, the centralised simulations guarantee that all systems relying on simulated values use the same simulation and will not differ due to different simulation lattices etc.

As the simulation can provide parameters that require time expensive measurements, such as beta functions and chromaticities, the online simulation can be used to optimise the regular operation conditions. For example, this can be used to optimise the sextupole magnets without having to redo the chromaticity measurements each time.

Since KARA is an accelerator test facility, often new operation modes and optics are implemented. By using the integrated simulation it is straightforward to test new optics before the actual beam time and reduces the time spent on optimisations. KARA offers its services to scientists of external institutes and as part of KIT as university also to students. For such users the tight integration and coherent interface to the rest of the control system offers a fast learning experience and removes the necessity of obtaining a simulation code, lattice and the knowledge how to operate it.

The code is setup in a way that multiple instances (with different prefixes) can be spawned to circumvent the problem of having only one simulation at a time for potentially multiple users. As an addition in the future, the individual instances could be lockable by users to avoid interference.

RESULTS & OUTLOOK

The system is successfully implemented and and first comparison measurements have been carried out. While the simulations can be very well used for studies and beamtime preparations, some simulated parameters do not perfectly match the values measured at the machine. As an example, a comparison for the tunes is shown in Table 1.

Table 1: Measured and Simulated Tunes

	Measured Tune	Simulated Tune
Horizontal	6.77	6.73
Vertical	2.81	2.94

The simulation lattice as well as the conversions between currents through the magnets and the magnet strengths should be improved. With the integration of the simulation into the control system and therefore direct availability algorithms that can help with those improvements can now be applied, possibly leveraging machine learning. Furthermore, the simulation uses perfect magnets without misalignments or fringe fields. The latter have been studied before in [7]. Also, the insertion devices are not implemented. Therefore, improvements to the simulation could be made by including the above mentioned effects and elements.

Improvements to the entire system could be made in the form of extensions of simulated features. That could include the simulation of emitted radiation or the inclusion of insertion devices in the lattice. Furthermore, an additional setting allowing for tracking simulation could be introduced to for e.g. estimations of beam lifetime or to identify critical apertures in the ring through loss simulations.

To overcome the centralisation issue with multiple users, the simulation could be put into a python package separate from the IOC control logic. This could allow for usage of the pre-setup simulation from within python scripts that offer almost the same convenience as the operation via the CSS panels.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Union's Horizon Europe research and innovation programme under grant agreement no. 101057511.

REFERENCES

[1] Eurolabs, https://web.infn.it/EURO-LABS/

[2] LatticeJSON, https://nobeam.github.io/latticejson/

[3] EPICS, https://epics-controls.org/

[4] I. Agapov *et al.*,"OCELOT: A software framework for synchrotron light source and FEL studies", *Nucl. Instrum. Meth. A*, vol. 768, pp. 151-156, 2014. doi:10.1016/j.nima.2014.09.057

[5] Control System Studio, https://controlsystemstudio.org/

[6] pythonSoftIOC, https://github.com/dls-controls/pythonSoftIOC

[7] M. Streichert *et al.*, "Simulations of fringe fields and multipoles for the ANKA storage ring bending magnets", in *Proc. IPAC'12*, New Orleans, LA, USA, May 2012, paper TUPPP011, pp. 1626–1628. https://jacow.org/ipac2012/papers/tuppp011.pdf

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP18

STATUS, RECENT DEVELOPMENTS AND PERSPECTIVE OF AVINE VIDEO SYSTEM

Stefan Weisse*, David Melkumyan, DESY, Zeuthen, Germany
Philip Duval, DESY, Hamburg, Germany

Abstract

DESY's TINE-powered Video System, originally released in 2002, was last presented in 2011 at ICALEPCS [1], at this time not yet known under the name Advanced Video and Imaging Network Environment (AVINE). AVINE provides a framework and toolkit for operators, physicists and technicians related to, but not limited to, Ethernet-based imaging at accelerator facilities. Over the past decade, the major emphasis was put on extended support, incorporating user requests, migrating to the latest Windows and Linux operating systems and the latest Java Virtual Machine, all while replacing legacy GigE Vision APIs in order to support past, current and future camera hardware. In this contribution, the current status, layout, recent developments and perspective of AVINE is described. The focus will be on experience migrating to future-oriented (still under vendor support) GigE Vision APIs, the recently upgraded image (sequence) file format, and first experiences on Windows 11.

OVERVIEW

The Video System explained further on originates at the Photo Injector Test Facility at DESY in Zeuthen (PITZ) [2]. Started in 2002 as a test facility at DESY for research and development on laser-driven electron sources for Free Electron Lasers (FEL) and linear colliders, PITZ has been extended over the last years in order to study different applications of photo injectors, e.g. for the generation of THz SASE light [3]. A significant milestone, first emission of THz light from the SASE FEL, has been reached in August 2022 [4].

Despite its to some extent different requirements, AVINE has been exported over the years to accelerator setups on DESY campus in Hamburg, namely PETRA III (e.g. machine diagnostics [5], user beamlines and so-called beam paths E-Weg and L-Weg), REGAE, SALOME and User Beamlines of Petra III at EMBL Hamburg. In addition, small installations have been realized over the years as well, e.g. monitoring and positioning of an electron beam at an electron welding machine in mechanics workshop and standalone AVINE video installations on notebooks for live video, data taking and analysis, which can also be used offline/off-site.

Ethernet-based industrial vision cameras which are conforming to standards GigE Vision and GenICam can easily be connected and used. In general, image acquisition is done with a repetition rate of 1 to 20 Hz, either initiated by external trigger signal (TTL pulse) or by an internal clock of the camera. Most cameras are greyscale (non-colour) with 8 to 12 bits per pixel. AVINE is integrated into the TINE control system [6]. Network transport of video image stream is done via TINE, either via TCP or multicast. To use available network bandwidth more effectively, JPEG compression can be applied on network transport. As a trade-off, greyscale pixel bit depth is limited to 8 bits when using JPEG.

Image size varies from hundreds of kilo- to several megapixels. Colour images (RGB 24 bits) are supported, however client GUIs and applications for measurements and analysis are mainly designed for greyscale data.

The Universal Slow Control System (USC) is designed to control various types of cameras used in the AVINE Video System, providing a common approach to configuring, displaying and controlling camera slow control settings required for PITZ operation, e.g. exposure time and analogue video signal gain. The USC system consists of a USC client, a USC server, and a USC Server Configurator implemented in Java. The USC Client is a TINE-based GUI client application that includes the functionality required by an operator to monitor and control the settings of all available cameras in the Video System. The USC Server is a TINE-based server that provides slow control of various types of cameras using various communication protocols (e.g. RS-232 for analogue cameras, TINE-based for controlling Gigabit Ethernet cameras via AVINE SGP servers). The USC Server Configurator (USCSC) is a GUI application for easy configuration of the USC Server, allowing server maintenance personnel to use predefined templates, modify them, or create a new template from scratch. It also checks server configurations and displays appropriate errors and troubleshooting tips to avoid unexpected USC server behaviour at runtime.

A versatile easy-to-use albeit aged Windows client application called Video Client 3 can be used for live view, data taking, measurements and on/off-line analysis.

With regards to platform independence, a JAVA component called TINE ACOP Video bean is available, which acts as a basis to create client-side GUIs with special functionality based on the experimental setup. The Video bean is also integrated into the TINE Instant Client, a basic Java GUI to browse, read and write TINE properties.

LAYOUT

The general layout of AVINE is shown in Fig. 1. AVINE is following a component (object-oriented) based approach. The idea is to hide implementation details and only expose what is necessary, so that changes in implementation are mainly kept inside a single component. For example, migration to a different API to interface cameras is only reflected in the front-end server (so-called Small Grabber Part (SGP)).

* stefan.weisse@desy.de

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP18

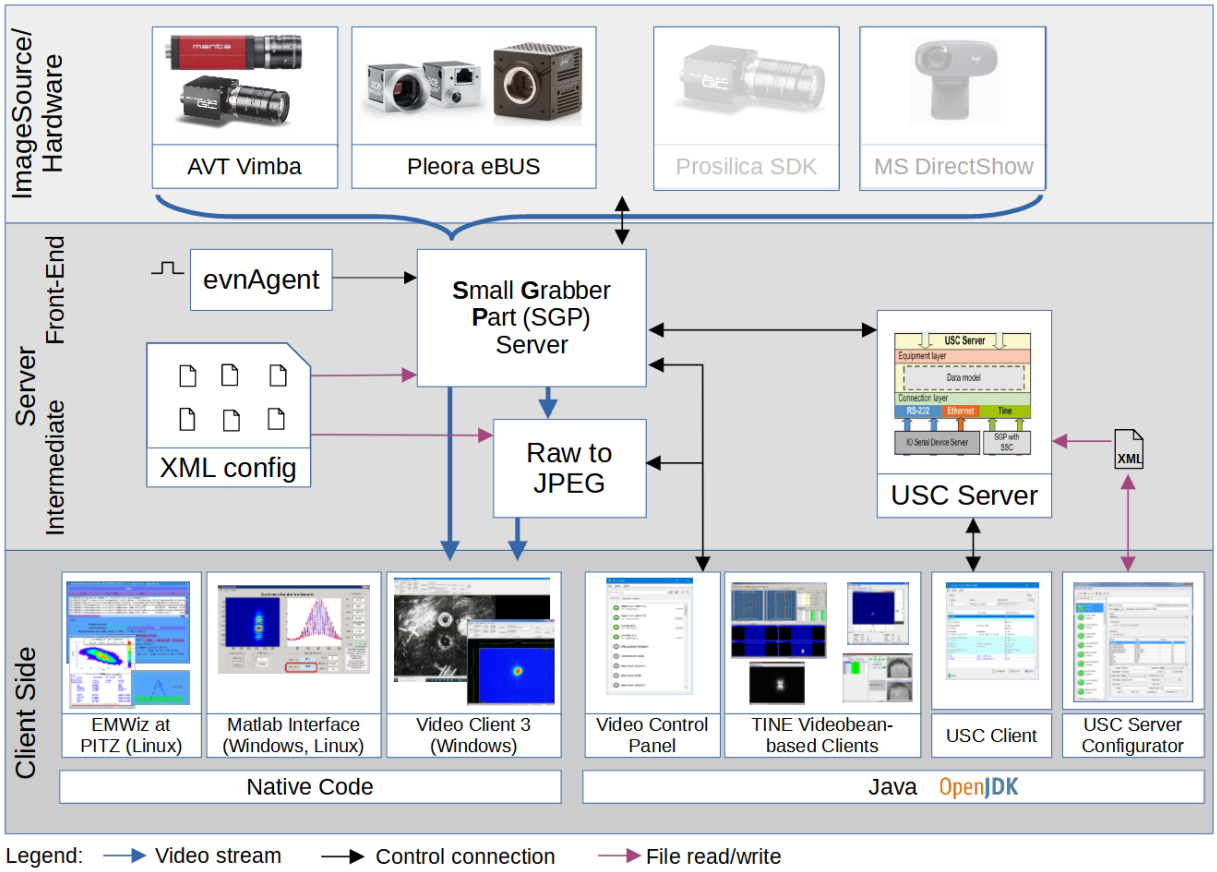


Figure 1: Simplified Layout of AVINE Components and their Interaction.

All other components continue to work as before, without need for modification. Server-side components running on the same computer can be interconnected efficiently using shared memory. From server to client side, the TINE protocol is utilized, using either TCP or UDP multicast. A dedicated well-defined image type is used to transport video images. Under increasing bandwidth demands and in general busier networks, multicasting video images via UDP proved to have stability issues. In the future, it is foreseen to change the default transport of video images to TCP where UDP Multicast is still used.

Server-side components running on the same computer can be interconnected efficiently using shared memory. From server to client side, the TINE protocol is utilized, using either TCP or UDP multicast. A dedicated well-defined image type is used to transport video images. Under increasing bandwidth demands and in general busier networks, multicasting video images via UDP proved to have stability issues. In the future, it is foreseen to change the default transport of video images to TCP where UDP Multicast is still used.

STATUS

All current production-level server-side and client-side installations are stable and running on monthly security patched Windows 10 (Server, LTSC, Enterprise, Pro) computers. On Linux (EMWiz, Matlab Interface), Red Hat Enterprise Linux based Scientific Linux 7 (SL7) is used. Ubuntu, being considered during implementation and testing, should work as well.

RECENT DEVELOPMENTS

As of April 2020, the JAI SDK and Control Tool, used to interface cameras of companies JAI and Basler, is no longer supported. In addition, PvAPI, widely used to interface Prosilica product line of cameras by vendor Allied Vision Technologies (AVT), has been superseded by AVT Vimba SDK over the past 10 years.

As a successor of JAI SDK for supporting modern JAI cameras (CMOS sensor), JAI recommends eBUS SDK by vendor Pleora. Test have shown that Pleora eBUS is flexible in terms of which cameras can be operated with it, so it was decided to be used as a successor for already existing AVINE camera installations where JAI SDK was used.

Currently, v6.1 of eBUS SDK is used. In contrast to previous APIs like JAI SDK and Prosilica PvAPI, eBUS SDK requires a paid license. A subscription license is required for developer support. A runtime license is in general required, too (one-time payment, bound to MAC address or USB dongle). This complicates development and maintenance. It would not have been chosen if there had been an easier option available. For some JAI cameras, no runtime license is required, because the license is part of the camera

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP18

firmware. Video servers that use Pleora eBUS as interface to cameras are already running in production.

To have a second API option for the many Prosilica cameras installed available, Vimba SDK is going to be used as a successor of Prosilica PvAPI. No extra license requirements, the lower cost and regular updates from the manufacturer's website (open access) eases development and use. Proof of concept tests using Vimba 4.2 and 6.0 have been finished successfully. Productive installation is foreseen, the Vimba-based video server needed for that is to be implemented in the coming months.

It must be noted that experience while developing and using eBUS SDK and Vimba SDK showed that the discontinued APIs were superior in terms of ease-of-use, complexity and flawlessness. This is not a good trend. Moreover, accessing Pleora eBUS SDK requires an account and login credentials. During a year of development subscription, no updates to the software were provided.

File Format IMC2

In order to store raw loss-less images, image sequences and background images, proprietary yet well-defined file formats had been designed in the years 2002 to 2005 and are still used today. For uncompressed storage, the file formats IMM (Image Multiple) and BKG (BacKGround image) support greyscale images with very limited metainformation to image: width, height, and a so-called scale factor (pixel to millimetre ratio for both width and height). In successive years, the compression of images was implemented to save disk space in file formats IMC (Images Compressed) and BKC (BacKground Compressed). In 2008 there was an attempt to supersede these file formats by using standardized file-formats PNG (widely used) and MNG, but in the end this attempt could not be completed.

In the meantime, detailed metainformation (e.g. scale factors, time stamp, event number, camera port name (location of camera) ...) are distributed with each image in a so-called image header when transferred via control system protocols TINE and DOOCS over the network at DESY.

Following a request to be able to store individual scale factors for x (width) and y (height), it was decided to implement a new file format for this. A feasible approach from the point of view of using widely adopted standards would be to put PNG files into a ZIP-file-like container, which could easily be viewed by standard tools. But having tight time constraints while aiming to keep the new file format as simple as possible and maintaining full open source policy, this idea cannot be implemented in a reasonable amount of time.

As an alternative, a new proprietary hence well-documented file format (IMage Compressed 2 (IMC2)) was designed, documented and implemented. A sketch of IMC2 file structure is shown in Fig. 2.

The new file format is going to store the detailed metainformation contained in the image header as human readable text (key=value) side-by-side with the compressed (zlib [7]) binary image data.

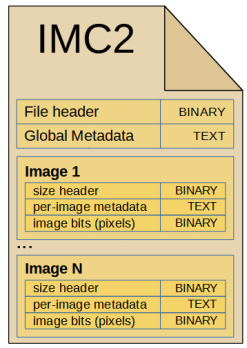


Figure 2: IMC2 File Format.

A release candidate of the file format has been designed and documented. The AVINE C library, available for 64-bit Windows and Linux, has been upgraded to provide functions to save and load the new file format. Based on the C library fundament, further integration into Matlab (mex files) and existing multipurpose application Video Client 3, is subject to happen in the next months.

Windows 11

As end of life of Windows 10 is only few years away, firsts tests of AVINE software have been done on Windows 11 (21H2, patch level 2022-08 and 2022-09). At DESY, a Windows 11 rollout is currently in the early stages, which limits the effectiveness and possibility of qualified tests. Nevertheless, tests of AVINE client and server software and of camera SDKs like eBUS and Vimba on Windows 11 look promising. No changes were necessary, one could simply continue to use existing software. But as time available for testing was short, it cannot be excluded that something was overlooked.

OUTLOOK

It is foreseen to continue supporting already existing camera installations of different camera vendors (AVT, Basler, JAI), e.g. at PITZ, PETRA III, REGAE and SALOME, possibly continuing support also at EMBL Hamburg. In the next months, priority will be the integration of new file format into Matlab and Video Client 3. Afterwards, implementation of Vimba SDK-based video server needs to be completed with the aim of discontinuing the use of PvAPI in the next years. Once ready, Allied Vision CMOS-based cameras Manta G-158B and G-319B, foreseen as replacement of aged Prosilica CCD cameras, can be tested under real accelerator conditions. Having only one productive use case at the moment, the video server which uses Microsoft DirectShow to get the video stream from hardware (Webcam) may be discontinued in future.

As the TINE control system protocol, used throughout AVINE for communication over the network, is planned to be abandoned in the next years, migration to DOOCS control system [8] will be attempted.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP18

REFERENCES

[1] S. Weisse, D. Melkumyan, and P. Duval, "Status, Recent Developments and Perspective of TINE-powered Video System, Release 3", in *Proc. ICALEPCS'11*, Grenoble, France, Oct. 2011, paper MOPMS033, pp. 405-408.

[2] F. Stephan, C.H. Boulware, M. Krasilnikov, J. Baehr, *et al.*, "Detailed characterization of electron sources yielding first demonstration of European X-ray Free-Electron Laser beam quality", *Phys. Rev. ST Accel. Beams*, vol. 13, p. 020704, 2010. doi:10.1103/PhysRevSTAB.13.020704

[3] P. Boonpornprasert, G. Georgiev, G. Koss, M. Krasilnikov, *et al.*, "Extension of the PITZ Facility for a Proof-of-Principle Experiment on THz SASE FEL", in *Proc. FEL'19*, Hamburg, Germany, Aug. 2019, pp. 38-40. doi:10.18429/JACoW-FEL2019-TUP001

[4] M. Krasilnikov *et al.*, "First Lasing of the THz SASE FEL at PITZ", presented at the FEL2022, Trieste, Italy, Aug. 2022, paper MOA08, to be published.

[5] A. I. Novokshonov, A. P. Potylitsyn, and G. Kube, "Two-Dimensional Synchrotron Radiation Interferometry at Petra III", in *Proc. IPAC'17*, Copenhagen, Denmark, May 2017, pp. 177-179. doi:10.18429/JACoW-IPAC2017-MOPAB042

[6] TINE (Three-fold Integrated Networking Environment), http://tine.desy.de

[7] zlib Compression Library, https://zlib.nett

[8] DOOCS (The Distributed Object-Oriented Control System Framework), http://doocs.desy.de

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

PYTHON BASED INTERFACE TO THE KARA LLRF SYSTEM

E. Blomley*, J. Gethmann, P. Schreiber, M. Schuh, W. Mexner, A. Mochihashi, A.-S. Müller
Karlsruhe Institute of Technology, Karlsruhe, Germany
S. Marsching, aquenos GmbH, Baden-Baden, Germany
D. Teytelmann, Dimtel Inc., San Jose, California, USA

Abstract

The Karlsruhe Research Accelerator (KARA) at the Karlsruhe Institute of Technology (KIT) is an electron storage ring and synchrotron radiation facility. The operation at KARA can be very flexible in terms of beam energy, optics, intensity, filling structure, and operation duration. Multiple digital low-level radio-frequency (LLRF) systems are in place to control the complex dynamics of the RF cavities required to keep the electron beam stable. Each LLRF system represents a well established closed system with its own set of control logic, state machine and feedback loops. This requires additional control logic to operate all stations together. In addition, during special operation modes at KARA, extra features such as well defined beam excitation are needed. This paper presents the implementation of a Python layer created to accommodate the complex set of options as well as an easy to use interface for the operator and the general control system.

RF SETUP OF KARA

KARA makes use of two RF stations controlled by one LLRF system each. Each RF station consist of a klystron which feeds two cavities utilizing a magic T with a fixed 3dB splitting ratio. In addition, the smaller booster synchrotron is powered by one cavity with an additional LLRF system. The electron beam in KARA is accumulated at 0.5 GeV with a 1 Hz injection cycle in the booster synchrotron. The accumulation period can typically last up to one hour. After the injection is finished, the beam energy of the storage ring is slowly increased to its final operating energy. Depending on the final energy, the energy ramp lasts up to three minutes. Therefore, in our common operation scheme, the demands of the RF system differ substantially between the booster synchrotron and the storage ring. The first operates on a fast, pre-defined and synchronized voltage ramp, whereas the approach for the storage ring is to explicitly set the RF voltage level to accommodate for the increase in beam energy as the energy slowly increases. Table 1 shows a summary of the RF system characteristics.

Table 1: RF System Characteristics

Parameter	Booster	Storage Ring
f_{RF} (MHz)	500	500
Voltage Range (kV)	5 - 25	300 - 1500
Energy Range (GeV)	0.05 - 0.5	0.5 - 2.5
Ramp Duration (s)	0.8	170
LLRF Stations	1	2
Cavities	1	4

Digital LLRF System

Modern, digital LLRF systems take care of operating the RF cavities by controlling the amplitude and phase of the RF waves, while also providing access to data points to read out signals, configure the relevant parameters and operate the RF station in general. At KARA the LLRF9 [1] from Dimtel was installed in 2016 with the booster synchrotron following in 2017. Each LLRF system has up to 9 RF inputs as well as the option for additional slow inputs to, for example, monitor the cavity vacuum levels. The main internal signal processing takes place on a field-programmable gate array (FPGA) surrounded by a Linux operating system providing slow access via the EPICS control system [2], which is also the main control system of all accelerators at IBPT. The EPICS interface allows access to around 600 process variables. This includes access to a stored ramping curve, captured waveforms of the fast inputs and an integrated network analyser, among other features. The system comes with a full set of graphical user-interface (GUI) panels (see Fig. 1).

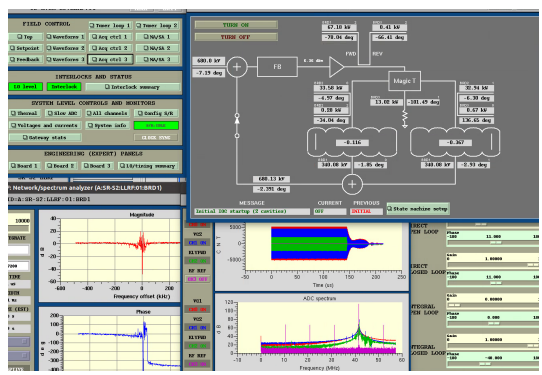


Figure 1: Native LLRF GUI. Visible is the top level control, the network analyser and the waveform acquisition tools.

USE CASES

The embedded EPICS controls and panels do an excellent job for what they were designed for, namely commissioning, configuring and monitoring the LLRF system. Still, there are several reasons as to why an additional software layer surrounding the core LLRF system might be useful.

Daily Operations

How the LLRF system is used in daily operations might be quite different from accelerator to accelerator or in this case even between the storage ring and the booster synchrotron.

* edmund.blomley@kit.edu

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

While the operation modes differ, the most common ones can typically be described by a fixed routine. A surrounding framework can directly implement these routines and therefore provide the operator with a simplified method to interact with and monitor the LLRF system, without requiring the extensive knowledge and training to work with the native interface. For example, in case of the booster synchrotron, the operator is mostly interested in making sure that the system is *ON* and operational without needing to manually:

- Reset interlocks if present
- Load a pre-defined ramping curve to waveform generator
- Switch on the LLRF feedback loop
- Switch to external hardware trigger and ramping mode

Control of Multiple LLRF Stations

Each LLRF system represents its own ecosystem, taking care of monitoring and controlling various feedback loops and external hardware such as motor controllers used to tune the cavities. But if multiple LLRF systems are in place, some form of high level control is desired. Certain actions just have to be mirrored to all stations, such as resetting interlocks and getting the units into an operational state. For other areas additional data points can be introduced, such as a voltage sum across all stations, which can be monitored and set without manual interaction with the individual LLRF systems. Another typical example is adjusting the relative phasing between the booster and the storage ring as a whole, as well as, between the two stations of the storage ring.

Extending Features

An additional software layer also creates an entry point to extend the capabilities of the LLRF system. An important feature for KARA is the internal waveform generator of each LLRF unit, which allows loading arbitrary waveforms of a maximum length of 512 elements. A useful extension of this is the ability to load a user-defined curve. Since it would be cumbersome to manually define 512 elements interpolation is useful, as well as dynamic scaling before the curve is loaded to the LLRF hardware. In addition, simple periodic patterns can be generated, such as sinusoidal or step functions which are useful for various beam studies [3–5].

Working with Physical Values

Devices in an accelerator are often controlled with properties using *hardware* units, such as Ampère *A* for magnet currents or amplitudes in *dB* for cavity fields. From an accelerator physics point of view, properties in *physical* units to monitor or even control elements of the storage ring are often desired. Taking the hardware units from the LLRF system and combining this information with additional data provided by the control system such a conversion can take place. Depending on the physical quantity, there might be a need to calibrate or estimate the actual value based on some additional model input. Typical examples are the ability to control the overall voltage level by setting a desired synchrotron frequency f_s (*calibrated*) or provide an estimation of the momentum compaction factor α_c (*model input*).

DESIGN CONSIDERATIONS

A custom software layer, internally referred to as *LLRF control service*, was developed with the original installation of the digital LLRF systems to provide most of the features discussed in the previous section. The service was written as a stand-alone C++ application. Since there were no comparable applications written in C++ at that point of time, additional code was required for the basic integration and interaction with our control- and operation systems. By now this system exists as an independent artifact without any other applications making use of the same C++ interface, making it difficult to maintain, improve or extend the code base. In addition, two separate versions of the control service exist, one for booster and the other one for storage ring operation, each having a different set of features. The different feature sets made sense at the time, but the demand on especially how the booster is operated by now has changed considerably. Table 2 lists the differences.

Table 2: Feature Disparity of Prior Framework

Feature	Booster	Storage Ring
Manual control possible	no	partially
Pre-defined ramping	yes	no
Energy based ramping	no	yes
Custom waveforms	no	yes

IMPLEMENTATION

Based on the considerations above and the criticality of the LLRF control service, where even small changes might have unintended but severe side-effects, the decision was made to fully reimplement this service from scratch. A key decision was made to switch to Python, which has seen widespread usage increase across our institute, from core accelerator services to machine learning applications [6].

Control System Integration

SoftIOC is a Python module which allows for creating an EPICS IOC which runs from within the Python interpreter supporting concurrency [7]. This allows creating an application which seamlessly integrates into our control system, as well as providing the option to include modules from the Python ecosystem (see Fig. 2).

Python Integration

The LLRF SoftIOC makes use of a set of modules referred to as the *KIT Accelerator Python Tools* [8]. These modules allow for an (internally) standardized and maintained access to common resources such as network storage, electronic logbook, data archive, system logging, measurement data handling, and shared routines across our other SoftIOCs.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

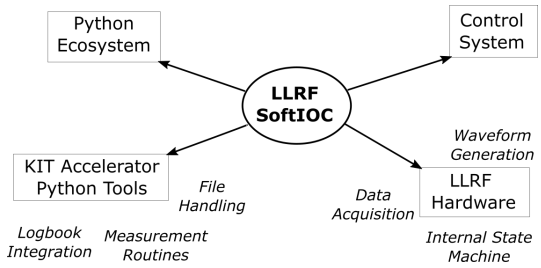


Figure 2: LLRF SoftIOC environment. SoftIOC allows seamless integration of our control system, the LLRF systems, our internal Python tools and access to the general Python ecosystem.

Access to other data points in the control system is provided using the *caproto* module [9].

Code Maintainability

To increase long term maintainability, common practices used in software engineering were applied, such as strict formatting rules and the requirement to document any public function, method and class. Tests are used to make sure that the behaviour of the code is kept consistent, also making it easier to implement potential changes in the future. Git in combination with a continuous integration workflow allows automatically running the tests as well as making sure that the code quality standards are being fulfilled before code can be added to the main code branch.

FEATURES

All features are available in both the booster and storage ring environment. In addition, each automation routine can be individually switched on or off, which allows the user to interact with the LLRF systems in varying degrees of semi- or fully manual control, if needed.

Event Logging

Certain events can automatically trigger the generation of a report for our electronic logbook [10]. For example, an interlock event during operation conditions causes the LLRF system interlock chain to be read out. Based on this information the software classifies the interlock as *internal* or *external*. In case of an internal interlock the waveforms of the RF signals are read-out and are attached as a plot to the report together with information regarding the current configuration of the LLRF system. Also, certain special routines during beam studies can create an entry to the logbook specifying the beam conditions and parameters used.

Simulator Mode

While tests of the Python code make sure that the functions behave as expected, the actual interaction with the LLRF hardware or the control system in general cannot be automatically tested. The simulator mode allows replacing either the process variables from the control system, such as the beam energy, or the process variables of the LLRF system. This is achieved by running a simulator SoftIOC which provides the necessary mock-up process variables as well as rerouting the process variables in the actual LLRF SoftIOC. In addition, the LLRF part can also be configured to use a LLRF spare unit if the actual hardware response is required.

State Machine Integration

With the updates of the LLRF system firmware, an internal state machine was added. Although this state machine cannot be used as a substitute for our typical operation modes, neither in the storage ring nor in the booster, it can take over interlock recovery, initial tuning and ramping down of the cavity fields. This allows removing a considerable amount of code and logic complexity, which was previously required for the initial start up of the LLRF system.

Measurement Routines

Thanks to having access to the Python ecosystem, including signal processing and fitting toolkits, more complex measurement or calibration routines can now directly be implemented in the LLRF SoftIOC code. One example is the calibration of the cavity voltage based on the measured synchrotron frequency f_s and subsequently a voltage scan to determine the momentum compaction factor α_c.

A more complex example is the implementation of precise phase or amplitude modulation of the cavity fields. This requires a measurement using the internal network analyser and a numerical solution based on the ratio of Bessel functions [11], whereas the user input only requires the desired modulation frequency and amplitude. To evaluate the modulation, a sequence for reading out the internal waveform is planned, which involves temporarily changing the acquisition engine from interlock detection to continuously measuring the field, taking a measurement, and switching back to interlock detection.

SUMMARY & STATUS

The dynamic range of operation modes at KARA creates a dynamic demand on our LLRF systems, from fully automated and transparent operation, to complex, semi-manual active measurements. Our approach for the LLRF SoftIOC represents a modular concept using Python, relying on robust implementations of independent modules from control system, file and logbook integration up to the inclusion of complex measurement routines with a strong focus on maintainability and extensibility. While not yet in production, first live tests in the booster are foreseen in the coming weeks.

REFERENCES

[1] LLRF9, https://dimtel.com/products/llrf9

[2] EPICS Control System, https://epics-controls.org/

[3] B. Kehrer *et al.*, "Time-Resolved Energy Spread Studies at the ANKA Storage Ring", in *Proc. IPAC2017*, Copenhagen, Denmark, May 2017, pp. 53–56. doi:10.18429/JACoW-IPAC2017-MOOCB1

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP20

[4] A. Mochihashi *et al.*, "Detuning Properties of RF Phase Modulation in the Electron Storage Ring KARA", pre-print: Nov. 2021. doi:10.48550/arXiv.2111.15555

[5] J. Steinmann *et al.*, "Increasing the Single-Bunch Instability Threshold by Bunch Splitting Due to RF Phase Modulation", in *Proc. IPAC'21*, Campinas, SP, Brazil, May 2021, pp. 3193–3196. doi:10.18429/JACoW-IPAC2021-WEPAB240

[6] P. Schreiber *et al.*, "Ocelot integration into KARA's control system", presented at PCaPAC'22, Prague, Czech Republic, paper THP16, this conference.

[7] pythonSoftIOC: *An EPICS IOC within the Python interpreter*, https://dls-controls.github.io/pythonSoftIOC

[8] J. Gethmann *et al.*, "Simple Python Interface to Facility-Specific Infrastructure", presented at PCaPAC'22, Prague, Czech Republic, paper THP17, this conference.

[9] caproto: *A pure-Python Channel Access protocol library*, https://github.com/caproto/caproto

[10] ELog, *Electronic Logbook package* https://elog.psi.ch/elog/

[11] D. Teytelman, "Phase modulation in storage-ring RF systems", e-print: Jul. & Sep. 2019. doi:10.48550/arXiv.1907.01381v2

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

USING REACT FOR WEB-BASED GRAPHICAL USER APPLICATIONS FOR ACCELERATOR CONTROLS

R. Bacher, J. Szczesny
Deutsches Elektronen-Synchrotron DESY, Germany

Abstract

Today, control applications need to run on a variety of different operating systems, including Windows, Linux, and Mac OS, but also Android and iOS. Programming languages like Java have tried to solve this problem in the past by providing a common runtime environment. However, this approach is insufficient or even unavailable for mobile devices such as tablets and smartphones. Another problem is the different form factors of mobile and desktop devices, which makes it difficult to develop portable applications. One way out of this dilemma is to use standard web technologies (HTML5, CSS3, and JavaScript) to implement applications that run in the browser, which is available for all platforms. Modern JavaScript web application frameworks combined with JavaScript graphics libraries such as D3 are suitable for building both very simple and very complex web-based graphical user applications. This paper reports on the status and issues that we encountered in our current developments with React.

LEGACY: WEB2CTOOLKIT

At DESY, the development of web-based user applications for accelerator operation started 2005 during the conversion of the high-energy booster synchrotron PETRAII into the synchrotron light source PETRAIII. In the following years, the so-called Web2cToolkit framework [1] was further implemented and its functionalities were extended step by step.

The Web2cToolkit is a collection of web services including

(1) *Web2c Synoptic Display Viewer*: Interactive synoptic live display to visualize and control accelerator or beam line equipment,

(2) *Web2c Archive Viewer*: Web form to request data from a control system archive storage and to display the retrieved data as a chart or table,

(3) *Web2c Messenger*: Interface to E-Mail, SMS and Twitter,

(4) *Web2c Logbook*: electronic logbook with auto-reporting capability,

(5) *Web2c Manager*: administrator's interface to configure and manage the toolkit,

(6) *Web2c Editor*: graphical editor to generate and configure synoptic displays, and

(7) *Web2c Gateway*: application programmer interface (HTTP-gateway) to all implemented control system interfaces.

Web2cToolkit is a framework for Web-based Rich Client Control System Applications. It provides a user-friendly look-and-feel and its usage does not require any specific programming skills. By design, the Web2cToolkit is platform independent. Its services are accessible through the HTTP protocol from every valid network address if not otherwise restricted. A secure single-sign-on user authentication and authorization procedure with encrypted password transmission is provided. Registered and so-called privileged users have more rights compared to ordinary users (read-only permission).

The Web 2.0 paradigms and technologies used include a Web server, a Web browser, HTML (HyperText Markup Language), CSS (Cascading Style Sheets) and AJAX (Asynchronous JavaScript And XML). The interactive graphical user interface pages are running in the client's Web browser. The interface is compatible with all major browser implementations including mobile versions. The Web2cToolkit services are provided by Java servlets running in the Web server's Java container. The communication between client and server is asynchronous. All third-party libraries used by the Web2cToolkit are open-source.

The Web2cToolkit provides interfaces to major accelerator and beam line control systems including TINE [2], DOOCS [3], EPICS [4] and TANGO [5] as well as STARS [6]. The toolkit is capable of receiving and processing JPEG-type video streams.

The Web2cToolkit is a proprietary framework that is not built on widely used JavaScript toolkits such as Dojo [7]. Due to progressing standardization efforts and the impressive development speed of modern web technologies, the Web2cToolkit is outdated and hardly maintainable in terms of performance and usability. Therefore, the development has recently been discontinued and only bug fixes are being made.

Figure 1: Hybrid app architecture [8].

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

Figure 2: Accelerator operations schedule.

PROGRESSIVE WEB APPS

Nowadays, users are spoiled by the powerful and rich features of graphical user applications that can be downloaded from Apple's or Google's application web stores. These applications tend to be native applications implemented for a single platform and without a common code base as they most likely originated from various platform-specific software development kits.

Hybrid applications (Fig. 1) offer a way out of this dilemma. Hybrid applications are multi-platform applications with a common code base. The code is based on platform-independent web technologies and a platform-specific component (WebView) wrapped into a native container (e.g. Cordova [9]). However, an application is still deployed and installed via a platform-specific application web store.

Progressive Web Apps (PWA) are platform-agnostic web applications developed using a set of specific technologies (e.g., HTML, CSS, JavaScript, WebAssembly, Service Worker) and standard patterns to take advantage of web and native application features. A PWA runs in the browser engine, either embedded in the browser window or as a fast-loading, network-independent standalone application added to the user's home screen. It looks and feels like a native app, has a responsive design, can be used on any device and includes offline storage and access to native features. A PWA can be discovered through a simple web search and is downloadable from a web server.

TECHNOLOGY EVALUATION

Modern web development platforms like Angular [10], React [11] or Vue.js [12] are well suited for the implementation of Progressive Web Apps. Recently, DESY has started to explore the possibilities of cross-platform graphical user apps based on React for the control of accelerators and beamlines. So far, 3 evaluation projects using the React framework have been carried out.

Web Data Display App

The Web Data Display app follows the design principles of the Web2c Synoptic Display Viewer application. It provides generic, customizable GUI widgets (e.g. Web2dPage, Web2dChart, Web2dTable, Web2dLabel-Value, …).

A specific graphical user application consists of a predefined set of graphical components whose configuration parameters are stored in a configuration file that is read and processed by the Web Data Display app at startup.

The graphical widgets are derived from the UI component set of the Ionic framework [13]. This open source framework is both platform-agnostic by using the cross-platform app runtime Capacitor [14] and SDK-agnostic by providing an interface to Angular, React and Vue.js. The Ionic framework provides ready-to-use build workflows for native, hybrid as well as Progressive Web Apps

Dashboard Apps

Two dashboard apps (accelerator operations schedule (Fig. 2), PETRAIII status display (Fig. 3)) based on React, JavaScript, HTML and CSS have been implemented to visualize the status and the performance of the operation of the DESY accelerators. These apps are created using the standard React project creation workflow. In most cases, style information is separated from the code and contained in CSS-files. The responsive behaviour is provided by the Material UI component library [15]. Routing within the apps is performed by the React Router library [16] which is not part of the React framework.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

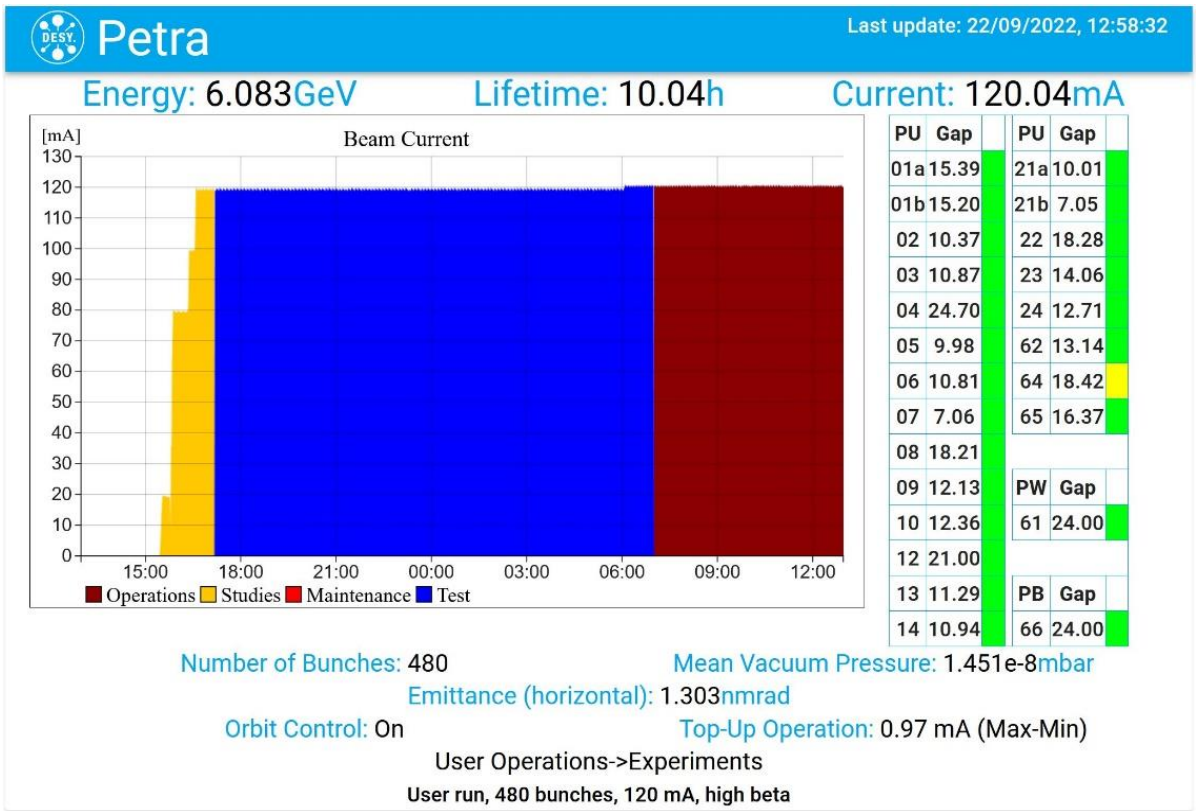


Figure 3: PETRAIII status display.

The apps communicate with the accelerator control system and receive their data asynchronously using the promise based axios [17] HTTP client from a RESTful web server which acts as a gateway to the accelerator control system. The data is encapsulated in a JSON structure that seamlessly integrates with JavaScript structures. Table 1 shows an example JSON data structure.

Table 1: Example JSON Data Structure

context	PETRA
server	VAC.ION_PUMP
device	SEK.ALL
property	P.MEAN
stsCode	0
timestamp	1663774206566
systemStamp	1873269314
fmtCode	5
numElements	1
format	FLOAT
status	success
data	1.3870215E-8

WebACOP Chart Library

The JavaScript WebACOP chart library is a novel variant of the so-called ACOP component family [18]. It provides a graphical component for displaying indexed data such as timeseries or histogram data. WebACOP wraps the D3.js [19] library in a JavaScript library. The D3.js library is suitable for creating dynamic, interactive data visualizations in web browsers and makes use of SVG, HTML5, and CSS standards. WebACOP dynamically configures the D3.js-based chart and injects data read asynchronously from the control system gateway server into the HTML/Canvas element of the drawing area. The WebACOP chart library is used to visualize live data from the operation of the PETRAIII accelerator as shown in Fig. 3. Its development is not yet completed.

FINDINGS AND CONCLUSIONS

The React framework is a powerful tool for rapid prototyping. Once implemented, components can be easily reused in other projects. While the implementation of simple components has proven to be relatively easy, the development of complex components can be quite time-consuming, cumbersome and error-prone.

The rapid release cycle of React versions often poses challenges for the developer, as backward compatibility is limited and incompatibilities with other third-part libraries can arise.

It turns out that the responsive design approach works well for dashboard applications, for example, but may not work for complex, well-designed control room applications with a variety of graphical widgets.

Our exploration of the possibilities of cross-platform graphical user apps based on React for the control of accelerators and beamlines is far from complete. For example, our experience in terms of debugging is still very limited and we also haven't explored how to migrate a React app into an Electron [20] desktop application.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-THP22

REFERENCES

[1] R. Bacher, "Light-Weight Web-based Control Applications with the Web2cToolkit", in *Proc. ICALEPCS'09*, Kobe, Japan, Oct. 2009, paper THP110, pp. 889-891.

[2] TINE, https://tine.desy.de

[3] DOOCS, https://doocs.desy.de

[4] EPICS, http://www.aps.anl.gov/epics

[5] TANGO, http://www.tango-controls.org

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

[7] Dojo Toolkit, https://dojotoolkit.org/

[8] Hybrid vs. Native, eBook, https://ionicframework.com/

[9] Apache Cordova, https://cordova.apache.org/

[10] Angular, https://www.angular.io/

[11] React, https://reactjs.org/

[12] Vue.js, https://vuejs.org/

[13] Ionic framework, https://ionicframework.com/

[14] Capacitor, https://capacitor.ionicframework.com/

[15] Material UI, https://mui.com/

[16] React Router, https://github.com/remix-run/react-router

[17] axios, https://github.com/axios/axios

[18] P. Duval, M. Lomperski, J. Szczesny, H. Wu, T. Kosuge, J. Bobnar, "ACOP. NET : Not Just Another GUI Builder", in *Proc. PCaPAC'18*, Hsinchu, Taiwan, Oct. 2018, pp. 139-143. doi:10.18429/JACoW-PCaPAC2018-THCB1

[19] D3.js, https://d3js.org/

[20] Electron, https://www.electronjs.org/

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR011

Taskomat & Taskolib: A VERSATILE, PROGRAMMABLE SEQUENCER FOR PROCESS AUTOMATION

L. Fröhlich*, O. Hensler, U. Jastrow, M. Walla, J. Wilgen
Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract

This contribution introduces the Taskolib library, a powerful framework for automating processes. Users can easily assemble sequences out of process steps, execute these sequences, and follow their progress. Individual steps are fully programmable in the lightweight Lua language. If desired, sequences can be enhanced with flow control via well-known constructs such as IF, WHILE, or TRY. The library is written in platform-independent C++ 17 and carries no dependency on any specific control system or communication framework. Instead, such dependencies are injected by client code; as an example, the integration with a DOOCS server and a graphical user interface is demonstrated.

INTRODUCTION

Like other scientific and industrial facilities, particle accelerators consist of many subsystems that need to be coordinated. The operation of any such complex system is inevitably governed by *processes* of various kinds. These processes can be implicit ("operators typically do it like this"), explicit (checklists and step-by-step instructions), or automated (implemented in hard- or software). Automated processes have several advantages over their manual counterparts:

- Faster execution
- Better reproducibility
- Reduced work load for operators
- Improved documentation of what happened when

The particle accelerator community has traditionally been in a good position to profit from automation in software because of its early adoption of distributed control systems. Tools to execute process steps automatically, so-called *sequencers*, have a long history [1–7]. Even today, this field keeps spawning new developments due to changing requirements and user expectations [8–10].

At DESY, the main tool for the automation of processes for more than a decade has been the aptly named *Sequencer/File Operator* [11]. It can execute linear sequences of steps with limited support for branches and jumps between the steps. Each step can write values to the control system or read them until certain conditions are fulfilled. Apart from this, the tool also provides save&restore functionalities for control system properties.

The *Sequencer* consists of a Java client with a graphical user interface (Fig. 1) and a Java server component that mainly provides central data storage and version control. Although the tool has proven its usefulness across practically all of DESY's accelerator facilities, several flaws have become apparent over the years:

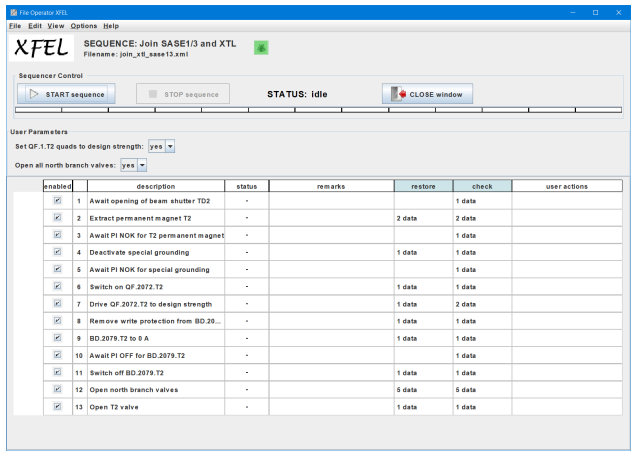


Figure 1: A procedure in the legacy DESY sequencer.

- The sequences are written as XML files with a poorly documented set of tags and attributes. Users typically have to interact with a Subversion repository to edit these files. This presents a high entry bar for creating or maintaining sequences, so that this task is left to very few experts and a lot of potential for automation is wasted.
- The kinds of sequences that can be expressed are limited: Control flow can only be directed through conditional *goto* statements, and there is no support for variables or arithmetic or string operations.
- The client-server communication is based on the TINE protocol [12] which has reached end-of-life and is slowly phased out.
- Sequences are executed on the client and require the graphical user interface. This prohibits their use from other components in the control system.

In combination, these shortcomings would be hard to overcome with incremental improvements to the existing code. Therefore, we have decided to develop a new sequencer from scratch.

Taskomat: DESIGN

The new Taskomat sequencer models processes as sequences of individual steps like its predecessor. In contrast to it, it is designed around the following goals:

- **Integration** into the control system: Sequences can be started, controlled, and manipulated from the control system.

* lars.froehlich@desy.de

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR011

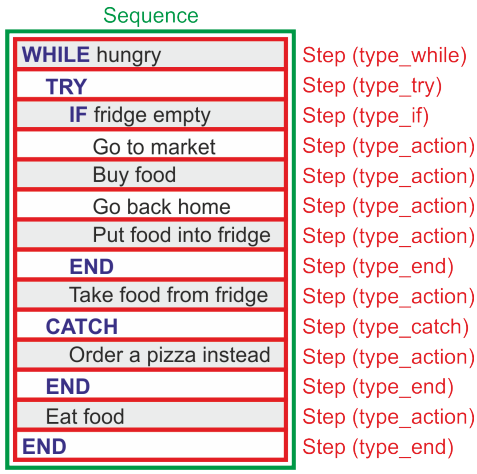


Figure 2: A process is modeled as a sequence of steps.

- **Control flow**: Sequences can contain well-known control flow constructs such as *if* or *while*, but no *goto*.
- **Programmability**: There is no inherent limit to the complexity of the procedures that can be expressed. Where necessary, users are able to program the desired functionality.
- **Ease of use**: The barriers for users to create, edit, and test sequences are as low as possible.
- **Separation** of control system dependent and independent code: As much of the code as possible has no dependency on the control system.

Figure 2 illustrates how a process is modeled as a sequence of steps: Typical steps just perform an *action*, but control flow steps like *if* or *while* can be used to express conditional execution or loops. Table 1 summarizes the available step types. In this high-level view, only the overarching control flow and user-defined descriptions of the individual steps are visible, providing a clear and unobstructed outline of the sequence.

What each individual step actually does is defined by a script written in the lightweight Lua extension language [13, 14]. Unlike most general-purpose programming languages, Lua has a clear and simple syntax that is easy to handle even for occasional users:

```
addr = "SOME/CONTROL/SYSTEM/ADDRESS"
for i = 1, 10 do
   print("Writing", i, "to", addr)
   dset(addr, i)
   sleep(0.5)
end
```

Here, Taskomat provides implementations of the functions *dset* (set a property in the control system) and *sleep*. Conditions for *if, elseif,* or *while* steps are also written in Lua. In these cases, the return value of the script determines if the corresponding branch is taken:

```
current = dget("ADDR/OF/A/CURRENT_MONITOR")
return current < 0.05
```

Table 1: Supported Step Types

Type	Description
action	Execute script
if	Conditional execution of block
elseif	Conditional execution of block
else	Conditional execution of block
while	Conditional loop execution
try	Execute block while intercepting errors
catch	Handle errors intercepted by try
end	Block-ending step

Individual steps are isolated from each other, but variables can be passed between them through explicit ex- or import from/to a *context* that is shared by the entire sequence.

IMPLEMENTATION

The Taskomat is implemented in standard C++ 17. The language was chosen for its platform independence and because bindings for all relevant control systems and for the Lua interpreter are available.

Taskolib Library

All of the control system independent code is gathered in a library that provides

- classes for modelling sequences and steps,
- a set of custom commands for Lua (e.g. *sleep* or *print* with output redirection),
- support for executing sequences asynchronously and aborting them at any time,
- support for receiving status updates and output from running sequences,
- support for serialization and deserialization of sequences.

The library includes a copy of Lua 5.4.3 and of the associated Sol3 C++ wrapper [15, 16] in version 3.2.3. Its only external dependency is the GUL14 [17, 18] basis library.

Taskomat DOOCS Server

So far, we have implemented one server for the DOOCS control system [19–21] that makes use of the library. It provides a thin layer of DOOCS properties to allow the inspection, modification, and execution of sequences. It also injects the DOOCS-specific commands *dget* and *dset* into the Lua interpreter to facilitate control system access from within the scripts. Most of the basic data types can already be read and written, which should make the server useful for DOOCS environments such as the European XFEL or FLASH facilities.

Graphical User Interface

We have created a simple graphical user interface with the *jddd* [22, 23] user interface builder. Figures 3 and 4 show screenshots of the main panel and of the step editor. This user interface blends in well with the jddd-based operation

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR011

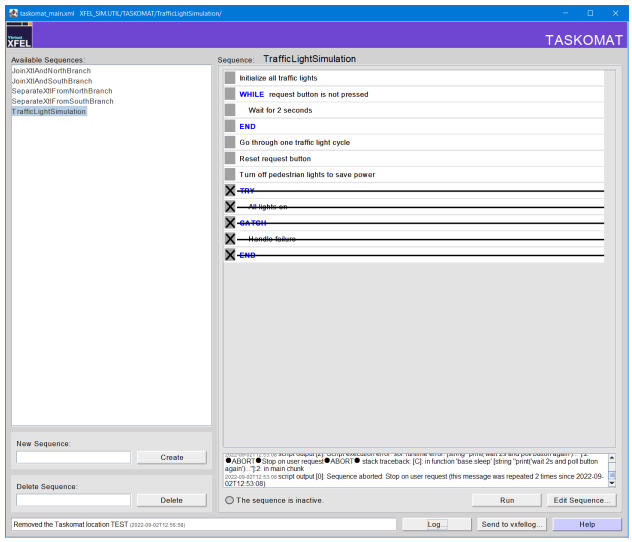


Figure 3: Main user interface for the Taskomat DOOCS server.

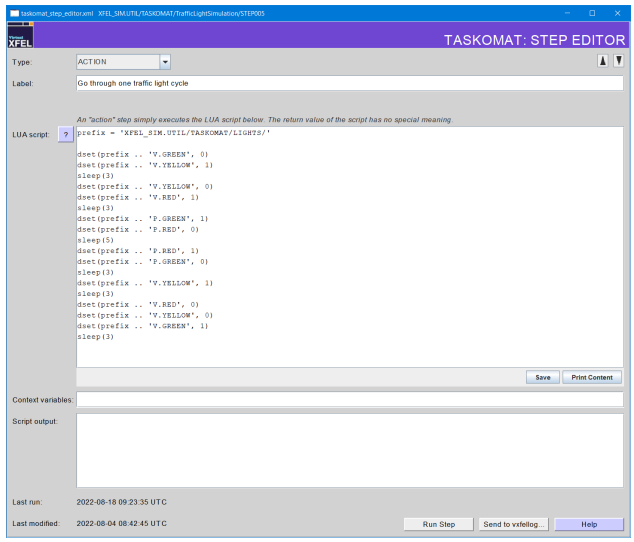


Figure 4: Step editor for the Taskomat DOOCS server.

panels of our accelerator facilities and should make it easy for operators and subsystem experts to develop their own procedures. It is, however, tightly coupled to the DOOCS server, and can therefore not be reused for developments with other control systems.

OPEN SOURCE

The Taskolib library is open source. The source code is published on GitHub [24] under the LGPL-2.1 [25] license. We welcome collaboration and external contributions. The source code of the Taskomat DOOCS server can also be made available on request.

STATUS AND OUTLOOK

The Taskolib library, the Taskomat DOOCS server, and the jddd-based user interface have reached "minimum viable product" state. Their feature set should be sufficient to perform useful tasks in our accelerator environments. We are now going to focus on the writing of sequences together with domain experts to gather feedback and experience with the software. The further development will depend on the outcome of this field test. We expect that features like automatic version control or improved nesting of sequences will be the next logical steps.

As we have seen with our DOOCS server, the effort of writing a Taskomat application for a specific control system is relatively low because the library contains most of the core functionality. This is not necessarily the case for the graphical user interface, though: In our case, the jddd panels were easy to create, but they cannot be reused for other control systems. The creation of a full-fledged GUI application could help with this. It would also make it easier to add special features like syntax highlighting for Lua.

ACKNOWLEDGEMENTS

The authors would like to thank Pedro Castro (DESY) and Giulio Gaio (Elettra–Sincrotrone Trieste) for many helpful discussions and ideas on sequencers and on the automation of accelerator operations.

REFERENCES

[1] P. Clout, M. Geib, and R. Westervelt, "Automation tools for accelerator control – A network based sequencer", in *Proc. LINAC 1990,* Albuquerque, USA, Sep. 1990, pp. 764–766.

[2] L. R. Dalesio, A. J. Kozubal, and M. R. Kraimer, "EPICS architecture", in *Proc. ICALEPCS 1991,* Tsukuba, Japan, Nov. 1991.

[3] A. J. Kozubal, D. M. Kerstiens, and R. M. Wright, "Experience with the State Notation Language and run-time sequencer", *Nucl. Instr. and Meth. A,* vol. 352, no. 1, pp. 411–414, Dec. 1994. doi:10.1016/0168-9002(94)91556-3

[4] J. A. Perlas, D. Beltrán, and J. Rosich, "Design and implementation of a finite state machine queuing tool for EPICS", in *Proc. ICALEPCS 1999,* Trieste, Italy, Oct. 1999, pp. 564–566.

[5] J. Patrick, "The Fermilab accelerator control system", in *Proc. ICAP 2006,* Chamonix, France, Oct. 2006, pp. 246–249.

[6] V. Baggiolini, R. Alemany-Fernandez, R. Gorbonosov, *et al.,* "A sequencer for the LHC era", in *Proc. ICALEPCS 2009,* Kobe, Japan, Oct. 2009, pp. 670–672.

[7] R. Alemany-Fernandez, V. Baggiolini, R. Gorbonosov, *et al.,* "The LHC sequencer", in *Proc. ICALEPCS 2011,* Grenoble, France, Oct. 2011, pp. 300–303.

[8] G. Gaio, P. Cinquegrana, S. Krecic, *et al.,* "A framework for high level machine automation based on behavior trees", in *Proc. ICALEPCS 2021,* Shanghai, China, Oct. 2021, pp. 534–539. doi:10.18429/JACoW-ICALEPCS2021-WEAL02

[9] W. Van Herck, B. Bauvir, and G. Ferro, "Automated operation of ITER", in *Proc. ICALEPCS 2021,* Shanghai, China, Oct.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR011

2021, pp. 628-630. doi:10.18429/JACoW-ICALEPCS2021-WEPV006

[10] D. Marcato, G. Arena, M. Bellato, *et al.,* "Pysmlib: A Python finite state machine library for EPICS", in *Proc. ICALEPCS 2021,* Shanghai, China, Oct. 2021, pp. 330–336. doi:10.18429/JACoW-ICALEPCS2021-TUBL05

[11] R. Bacher, "Commissioning of the new pre-accelerator control system at DESY", in *Proc. PCaPAC 2008,* Ljubljana, Slovenia, Oct. 2008, pp. 171-173.

[12] P. K. Bartkiewicz and P. Duval, "TINE as an accelerator control system at DESY", *Meas. Sci. Technol.,* vol. 18, pp. 2379–2386, July 2007. doi:10.1088/0957-0233/18/8/012

[13] R. Ierusalimschy, L. H. de Figueiredo, and W. Celes, "Lua – an extensible extension language", *Software: Practice & Experience* vol. 26, no. 6, pp. 635–652, June 1996. doi:10.1002/(SICI)1097-024X(199606)26:6<635::AID-SPE26>3.0.CO;2-P

[14] Lua website, https://www.lua.org

[15] Sol3 documentation website, https://sol2.readthedocs.io/en/latest/

[16] Sol3 source code repository, https://github.com/ThePhD/sol2

[17] GUL14 (General Utility Library for C++14) documentation website, https://gul14.info

[18] GUL14 (General Utility Library for C++14) source code repository, https://github.com/gul-cpp/gul14/

[19] O. Hensler and K. Rehlich, "DOOCS: A distributed object oriented control system", in *Proc. XV Workshop on Charged Particle Accelerators,* Protvino, Russia, 1996.

[20] L. Fröhlich, A. Aghababyan, S. Grunewald, *et al.,* "The evolution of the DOOCS C++ code base", *Proc. ICALEPCS 2021,* Shanghai, China, Oct. 2021, pp. 188–192. doi:10.18429/JACoW-ICALEPCS2021-MOPV027

[21] DOOCS website, https://doocs.desy.de

[22] E. Sombrowski, A. Petrosyan, K. Rehlich, *et al.,* "jddd: A tool for operators and experts to design control system panels", in *Proc. ICALEPCS 2013*, San Francisco, USA, Oct. 2013, pp. 544–546.

[23] jddd (Java DOOCS Data Display) documentation website, https://jddd.desy.de

[24] Taskolib source code repository, https://github.com/taskolib/taskolib

[25] GNU Lesser General Public License version 2.1, https://www.gnu.org/licenses/old-licenses/lgpl-2.1.html

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR012

AUTOPARAM, A GENERIC ASYN PORT DRIVER WITH DYNAMIC PARAMETERS

Jure Varlec*, Cosylab d.d., Ljubljana, Slovenia

Abstract

Implementing EPICS device support for a specific device can be tricky; implementing generic device support that can integrate different kinds of devices sharing a common interface is trickier still. Yet such a driver can save a lot of time down the road. A well-known example is the Modbus EPICS module: the same support module can be used to integrate any device that speaks the Modbus protocol. It is up to the EPICS database to map device registers to EPICS records. Because no changes to the driver code are needed to integrate a device, a lot of effort is saved. At Cosylab, we often encounter device controllers that speak bespoke protocols. To facilitate development of generic drivers, we wrote the Autoparam EPICS module. It is a base class derived from `asynPortDriver` that handles low-level details that are common to all generic drivers: it creates handles for device data based on information provided in EPICS records and provides facilities for handling hardware interrupts. Moreover, it strives to provide a more ergonomic API for handling device functions than vanilla `asynPortDriver`.

INTRODUCTION

When it comes to putting together a control system, one of the advantages of EPICS is that, given existing device support, integrating devices requires little or no programming: data that devices provide or consume is mapped to process variables *declaratively* through *records* in an EPICS database [1]. Because EPICS is not merely software, but a vibrant community, lots of existing device support modules are readily available [2]. Chances are, then, that integrating a widely-used type of device into your control system requires very little effort.

Of course, not all devices are widely used. Some are brand new, some are niche, some may even be developed specifically for a particular machine. In such a case, developing a new device support layer cannot be avoided. This is an arduous task: device support represents a mapping from device functionality to records. This means that a device support layer is required for *each record type* of interest [3]. Much of this work is repetitive and results in pretty much the same record-specific code across many different device types. For this reason, the device support layer of EPICS lends itself well to encapsulating the repetitive common code in a reusable module.

`asynDriver` [4] is such a module. It has become the go-to module for integrating new devices, and even whole data processing pipelines [5]. Instead of implementing EPICS per-record device support directly, one instead implements one or more `asyn` *interfaces* to wrap the low-level device driver or communication protocol. These interfaces are then used by `asyn`'s *generic* per-record device support layer which covers pretty much all record types that are meant to get data into or out of hardware devices. By using `asyn`, one is thus saved from a fair amount of repetitive and error-prone work: write one driver, support all records automatically. The most straightforward way to create a device support module based on `asyn` is to create a C++ class derived from the `asynPortDriver` base class.

One can go a step further and recognize that some devices are themselves "generic" in the sense that they are merely front-ends to other devices. One example are programmable logic controllers (PLC) which often serve as interfaces to sensors and actuators. It is thus natural to desire an EPICS module that would communicate with a particular type of PLC in a generic enough manner to facilitate integrating it into the control system regardless of which peripherals are connected to it. There are, in fact, several such EPICS modules available, the Modbus module [6] being a very nice example. It allows integration of any PLC (or other device) that speaks the Modbus protocol. With a device support as generic as this, the "no programming" ideal mentioned in the beginning is truly possible: the EPICS database defines the mapping between Modbus registers and records, interpreting and giving meaning to the raw data values contained in the registers. Regardless of which peripherals are connected to the PLC, no changes to the drivers are needed.

But the Modbus protocol is just one possibility, chosen as an example because it is so widely known and used. As with all devices and protocols, there are many generic ones that are niche. Yet even for niche protocols, it makes sense to implement a module as generic as the Modbus module. A protocol may only be used at a single facility, yet that may still mean that it is used in many different setups (e.g. with different peripherals).

At Cosylab, we have helped with EPICS device support for such generic devices many times now. We felt the pain of doing the same kind of work several times. We have thus decided to develop a C++ base class which encapsulates the common functionality that needs to be implemented by any generic device support code; most importantly, this includes generating per-record handles dynamically based on information provided in EPICS records. This C++ base class builds on top of `asynPortDriver`, allowing us to reuse the basic functionality provided by this `asyn` class, and, crucially, to reuse the `asyn` per-record device support layer. We found this module, called `autoparamDriver` [7], to be generally useful, and are releasing it to the community.

* jure.varlec@cosylab.com

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR012

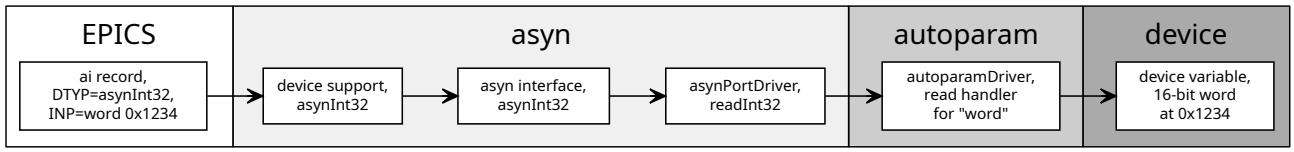


Figure 1: An example of reading from a device, showing the layers involved, starting from an EPICS record. Note how the DTYP field of the record specifies which `asyn` calls will be involved, while the INP field provides data for `autoparamDriver`. Specifically, `word` refers to a device function that returns a 16-bit word; `autoparamDriver` uses this in its implementation of `asynPortDriver::readInt32()` to find a read handler registered by a subclass. The other part of INP, `0x1234`, is used by the handler for `word` as an address on the device.

GOALS

`autoparamDriver` has several goals, listed here in order of priority.

1. *Allow the EPICS database to specify the mapping between process variables and device variables* (e.g. device registers) without modifying the driver or device support code.

2. *Provide approachable documentation.* `asyn` documentation, is quite general and not very EPICS-oriented.[1] The intent is to complement it with a gentler introduction to the concepts involved in writing a driver.

3. *Reuse as many of `asyn`'s facilities as possible and sensible.* The intention is to save development time[2] at the cost of conforming to certain constraints imposed by the `asynPortDriver` base class.

4. *Support different paradigms of processing `I/O Intr` records.* Some examples: processing based on hardware interrupts, from a periodic scan thread in the driver, and on writing to a register from another record. `asyn` does a good job at this and `autoparamDriver` should make this easier if possible, but not more difficult.

5. Related to the goal of providing approachable documentation, *provide a more ergonomic API than bare `asynPortDriver` when possible*. Internal details of `asyn` leak through its APIs, along with concepts that are hard to grasp or are heavily overloaded. A good example of both is the concept of "reason": reasons of different kinds are passed to `asynPortDriver` functions as parameters, but can mean different things depending on context. This can be difficult to follow even for an experienced developer.

DESIGN

The `autoparamDriver` EPICS module provides the `Autoparam::Driver` class, which is derived from `asynPortDriver`, and, in turn, needs to be subclassed and extended with functionality needed to talk to a specific type of device. There are two ways to look at such a driver: the end-user point of view and the driver developer point of view.

From the point of view of the end user, who is tasked with setting up an EPICS input-output controller (IOC) and its database for their particular situation, it all starts with the EPICS database (see goal 1 above). The database provides the information needed to transfer data to or from the device, and the layers underneath take care of it. This is shown in Fig. 1 where a "word" at address 0x1234 is requested from the device. The user needs to know

- which `asyn` device support layer is needed,
- which "device function" they want (e.g. `word`),
- and which parameters this function needs (e.g. `0x1234`).

That a particular device support needs to be chosen explicitly may seem superfluous, especially because `autoparamDriver` ties it to the device function (as we will see later). However, this is how EPICS records work, and for good reason.[3]

The driver developer, on the other hand, has much more freedom in how to approach the problem. There are three issues to tackle:

- opening "channels" or "handles" to the device based on which records are instantiated in the database,
- handling read and write requests from records,
- and pushing data to `I/O Intr` records, i.e. records that are to be updated in response to events originating on the device.

In the rest of this section, we will take a look at how `autoparamDriver` approaches them and what needs to be provided by the subclass of `Autoparam::Driver` and its ancillary classes to make things work. To make things easier to follow, the APPENDIX has an abbreviated listing of the `Driver` class.

[1] Excepting, naturally, the chapter on EPICS device support.

[2] This goal had high priority during the development of `autoparamDriver` because other in-development drivers depended on it. Its importance will diminish with time.

[3] A record can change its behavior significantly based on which device support is in use. For example, `asyn` allows an `ai` record to be used both with `asynInt32` and `asynFloat64` interfaces which handle unit conversions differently.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR012

Device Variables

When an EPICS records that uses `asyn` device support is initialized, it registers a handle for itself by making a call that ends up executing `asynPortDriver::drvUserCreate()`. This function is overridden by `autoparamDriver` and is the point where handles to device data are instantiated. These handles are called *device variables* as that is what they represent. Each variable has an associated *device address*. Based on the string provided in the record's INP field, which is passed to `drvUserCreate()`, a variable is created in three steps:

1. The first (and possibly only) word of the input string is taken as a *device function*. We only proceed if a read and/or write handler has been registered for this function (as described later).

2. The rest of the input string is passed as-is to the function `parseDeviceAddress()` which is implemented by the subclassed driver. This function returns a `DeviceAddress` object.

3. If this address was encountered before,[4] the existing `DeviceVariable` object is returned. Otherwise, the `createDeviceVariable()` function implemented by the subclassed driver is called to create a new object.

The `DeviceAddress` and `DeviceVariable` are abstract classes. They are meant to be subclassed so that they contain data relevant to the device.

Handlers

`Autoparam::Driver` overrides `asynPortDriver`'s read and write functions. These functions receive an `asynUser` pointer as a handle to the record that called them, and this handle contains a "reason" which can be used to determine which data is requested. Thus, an implementation of a read or write function typically contains a `switch` statement or an `if-else` ladder to dispatch on the "reason".

`autoparamDriver` provides that level of dispatch itself by recognizing that generic drivers commonly refer to a *device function*, such as the `word` function in the example shown in Fig. 1. `autoparamDriver` will examine the provided `asynUser` handle to find the associated `DeviceVariable`, then it will find the handler registered for the function associated with the `DeviceVariable` and call it. The subclassed driver thus needs to implement handlers for each device function it knows, and register these handlers in its constructor.

Read and write handlers are static functions[5] that take a `DeviceVariable` reference as an argument. They can cast it to whichever subclass the driver actually uses, thus getting access to everything the handler needs to communicate with the device. As an example, the signature of a read handler for 16-bit words looks like this:

```
Result<epicsInt32> readWord(DeviceVariable &var);
```

Note that the value read from the device is returned as part of the `Result`. This object also contains the error status and, if required, can override the alarm status and severity of the EPICS record that made the call. The type of data that it passes to `asyn` is given in its template argument. There is no mention of the 16-bit data type that is used to talk to the device: that is an implementation detail of the subclassed driver.

When the subclassed driver registers a handler that uses `epicsInt32` data type on the `asyn` side, the `Autoparam::Driver` base class knows that the data will move through the `asynInt32` interface and that this handler is a candidate when `asynPortDriver::readInt32()` is called. This is an example of a pattern that pervades `autoparamDriver`: instead of specifying the interfaces with an `enum` or having them as part of function names as is normally done in `asyn`, they are implied by the data types used. Scalars are passed as `epicsInt32` or `epicsFloat64`, digital IO uses `epicsUInt32`, arrays are wrapped in `Array<T>` and strings are wrapped in `Octet` (which is an `Array<char>`). There is a 1:1 mapping between data types and `asyn` interfaces.

Interrupts

`I/O Intr` records are processed via the mechanism provided by `asynPortDriver`, which is built on top of *asyn parameters*. Each `DeviceVariable` is backed by an `asyn` parameter, which is dynamically created during the call to `drvUserCreate()`; this is where the name of `autoparamDriver` comes from.[6] This means that processing `I/O Intr` records is done the same way as with plain `asynPortDriver`:

- set one or more scalars using `setParam()`, then call `asynPortDriver::callParamCallbacks()`;
- for arrays, use the `doCallbacksArray()`.

For scalars, `asyn` parameters take care of determining whether the value of a parameter has been changed and whether (and which) records needs to be processed.

`setParam()` and `doCallbacksArray()` are not *quite* the same as `asynPortDriver`'s `set*Param()` and `doCallbacks*Array()` family of functions. First, they take a reference to a `DeviceVariable` as the first argument. Second, they are *templates*. This continues the pattern noted earlier: `autoparamDriver` uses data types to determine which `asyn` interface to use. Thus, passing an `epicsInt32` to `setParam()` will dispatch to `asynPortDriver::setIntegerParam()` and the programmer does not need to deal with `asyn`'s naming idiosyncrasies (goal 5).

[4] Several records can refer to the same device variable.

[5] One of the constraints for `autoparamDriver` was that it has to conform to the C++03 standard, which constrains the design significantly, but allows for wider adoption.

[6] In truth, the name will stay appropriate even if this mechanism is replaced in the future, the basic design will remain the same. And it may well be replaced: reusing `asyn` parameters is in line with goal 3, but constrains `autoparamDriver` so that each device function can only be bound to a *single* interface. This was found to not be very limiting, but it may still be desireable to lift this restriction at some point.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR012

In accordance with goal 4, `autoparamDriver` allows one to process `I/O Intr` records

- during or after running write or read handlers,
- in response to hardware interrupts (e.g. from a callback function),
- or at any other time, in particular from a background scanning thread.

For each of these cases, there is a feature that supports it.

The code that calls read and write handlers can optionally also process `I/O Intr` records that are bound to the same device variable. A handler decides whether this is desired by setting a field in the `Result` structure it returns.

For a background scanning thread, it may be convenient to know which device variables have `I/O Intr` records that are interested in updates in a given moment. The `getInterruptVariables()` function returns a list of such variables.

For some devices, hardware interrupts need to be explicitly enabled, or a subscription needs to be set up for each device variable of interest. When a driver is registering handlers, it may also register an *interrupt registrar* function. That function is called (a) when the first record's SCAN field is switched to `I/O Intr`; (b) when the last record is switched away from `I/O Intr`. In other words, no matter how many records are bound to a particular device variable and have SCAN set to `I/O Intr`, only one call to the registrar will be done. The registrar function can then set up (or tear down) the subscription to hardware interrupts.

CONCLUSION

Let us summarize how the design of `autoparamDriver` supports its goals.

The primary objective is allowing the driver author to create a generic driver; such a driver facilitates binding device variables to records without modifying and recompiling the driver. To this end, `autoparamDriver` provides a mechanism to instantiate `DeviceVariable` objects that the subclassed driver can use as it sees fit. Parsing of device addresses that are entered into the EPICS records is left to the subclassed driver, allowing lots of freedom. Only the first word is interpreted as a *device function*, allowing the subclassed driver to be designed in terms of *function handlers*. This reduces the amount of necessary boilerplate.

`autoparamDriver` relies on the `asynPortDriver`'s *parameters* to help with processing interrupts, which constrains the design a bit, but builds on top of a well-tested foundation. `asynPortDriver`'s facilities also allow interrupts to be processed in a number of different scenarios.

While we recognize that API design is, to an extent, a matter of taste, we believe that `autoparamDriver` provides a more ergonomic facade over `asynPortDriver`'s functions by using templates instead of separate functions. The use of handlers instead of overloading read and write functions also furthers this goal. More importantly, `autoparamDriver` eschews various "reasons" that `asyn` uses, using instead `DeviceVariable` objects as handles to device data.

As for documentation [7], we aim to have not only a complete reference document, but also a gentle introduction to the concepts involved. Importantly, we do *not* provide an example driver. Instead, we provide a *tutorial*, each step accompanied with a short discussion of what needs to be considered. We hope that this will reduce cargo-culting and increase the quality of drivers based on `autoparamDriver`.

APPENDIX

Abbreviated[7] public interface of `Autoparam::Driver`.

```
namespace Autoparam {
class Driver : public asynPortDriver {
public:
    explicit Driver(const char *portName, DriverOpts const &params);
    virtual ~Driver();

protected:
    virtual DeviceAddress *parseDeviceAddress(std::string const &function,
                                              std::string const &arguments) = 0;

    virtual DeviceVariable *createDeviceVariable(DeviceVariable *baseVar) = 0;

    template <typename T>
    void registerHandlers(std::string const &function,
                          typename Handlers<T>::ReadHandler reader,
                          typename Handlers<T>::WriteHandler writer,
                          InterruptRegistrar intrRegistrar);

    template <typename T>
    asynStatus doCallbacksArray(DeviceVariable const &var, Array<T> &value,
                                asynStatus status = asynSuccess,
                                int alarmStatus = epicsAlarmNone,
                                int alarmSeverity = epicsSevNone);

    template <typename T>
    asynStatus setParam(DeviceVariable const &var, T value,
                        asynStatus status = asynSuccess,
                        int alarmStatus = epicsAlarmNone,
                        int alarmSeverity = epicsSevNone);

    std::vector<DeviceVariable *> getAllVariables();

    std::vector<DeviceVariable *> getInterruptVariables();
};
}
```

REFERENCES

[1] *EPICS database concepts*, retrieved August 2022. https://docs.epics-controls.org/en/latest/guides/EPICS_Process_Database_Concepts.html

[2] *EPICS hardware support database*, retrieved August 2022. https://epics-controls.org/resources-and-support/modules/hardware-support/

[3] *EPICS application developer's guide*, retrieved August 2022. https://docs.epics-controls.org/en/latest/appdevguide/AppDevGuide.html

[4] *asynDriver*, retrieved August 2022. https://epics-modules.github.io/master/asyn/

[5] *AreaDetector*, retrieved August 2022. https://areadetector.github.io/

[6] *The modbus module*, retrieved August 2022. https://epics-modbus.readthedocs.io/

[7] *The autoparamDriver documentation*, retrieved September 2022. https://epics.cosylab.com/documentation/autoparamDriver/

[7] Functions from the part of the public interface of `asynPortDriver` that are used by `asyn` must be public, but are not intended to be overriden further and are ommited here. See main text for the description and example signature of a handler, and short descriptions of some of the ancillary classes.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR013

PROGRESSION TOWARDS ADAPTABILITY IN THE PLC LIBRARY AT THE EuXFEL

T. Freyermuth*, S. T. Huynh†, B. Baranasic, M. Bueno, N. Coppola, G. Giovanetti, S. Hauf, N. Jardón Bueno, N. Mashayekh, A. Silenzi, M. Stupar, M. Teichmann, J. Tolkiehn, L. Feltrin Zanellatto, P. Gessler, EuXFEL, Schenefeld, Germany

Abstract

In 2011, the European X-Ray Free Electron Laser (EuXFEL) commenced parallel developments of their control system (Karabo) and the Programmable Logic Controller (PLC) library. The PLC library was designed to control basic beamline components and under as set of initial assumptions, the automation component was deferred to the control system layer. After five years of operation, it can be seen that not all initial assumptions scaled well to the operational needs of the facility resulting in limitations hindering progress. Having identified the issues, the PLC development is now focused on providing a more cohesive and adaptable solution. In utilizing the IEC61131-3 (3rd edition) features, the PLC library has been restructured towards a layered architecture with loose coupling between function blocks. The ultimate goal is to achieve a PLC library which is not only test driven and capable of quickly integrating in new devices, but can achieve dynamic linking not only between hardware and software, but also across software devices, aiding the rapid development of more complex hardware integration and higher-level automation.

INTRODUCTION

Programmable Logic Controllers (PLCs) have been historically used to provide control and automation of larger systems, and this is no different at the European XFEL (EuXFEL)[1]. To further optimise the generation of PLC projects, a library has been developed to provide buildings blocks, which are pieced together to form the backbone of a PLC project. While initially developed to reduce development time for PLC projects, the PLC library also incorporates a bespoke communication protocol and provides functionality which over time became less efficient and harder to expand upon. As such, a new version of the library is underway, addressing some of the short-comings of the original:

- A strictly layered architecture that is testable
- Dynamic linking is provided by the TwinCAT Hardware Abstraction Layer (TcHAL)
- An enhanced communication protocol
- Simpler inter-device and inter-plc communication
- Additional tools

Paving the way for dynamic, adjustable and configurable complex automation processes, which can easily be developed and tested, whilst leaving room for future changes in a completely disentangled manner.

* Tobias.Freyermuth@xfel.eu
† Sylvia.Huynh@xfel.eu

CURRENT LIBRARY

The current PLC library is built up of a collection of software representations of hardware devices, which are known as softdevices. During early development, it was a priority to ensure that the rapid deployment milestones of PLC projects were met. The softdevice building blocks aided quick delivery through device instantiation, ensuring conformity amongst hardware devices performing a similar function set, and creating a simplified way with how these devices interact with the Supervisory Control and Data Acquisition (SCADA) system known as Karabo [2]. Using a TCP/IP communication protocol developed in-house, a connection header entailing basic device class information known as a self-description provided the SCADA system with a means to obtain information to aid User Interaction (UI). As the number of PLC projects grew in size, it became relevant to ensure a means for peer-to-peer communication between softdevices, in addition to basic equipment protection. This was achieved through shared variables and interlocks - a set of conditions which when met, would trigger a set of actions. Lastly, an additional tool called the PLC Management System (PLCMS)[3] was developed to aid the automatic generation of PLC projects with linking between the hardware Input and Output (I/O) and the previously defined softdevices. The structure of the current library can be seen in Fig. 1.

FUTURE LIBRARY

Building on from the lessons learned thus far in regards to the existing PLC library, this section will explore the changes planned for the next version of the EuXFEL PLC library, and how they work and will be integrated.

PLC Library Goals

Whilst the current library is functional, it is hoped that the many limitations that are currently imposed due to historical design decisions can be overcome in the next version. As many of the previous requirements remain valid, this section details on how they can be enhanced or evolved. Additionally, new requirements, which will facilitate to provide better building blocks for future PLC projects are introduced.

A Complete Self-description of the PLC A highly important feature of the current library is the self descriptive nature of the PLC. When a connection is made to the PLC (via a TCP client), the PLC will send a description of all the devices currently on it. While incredibly useful, the self-description is still lacking some features for true com-

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR013

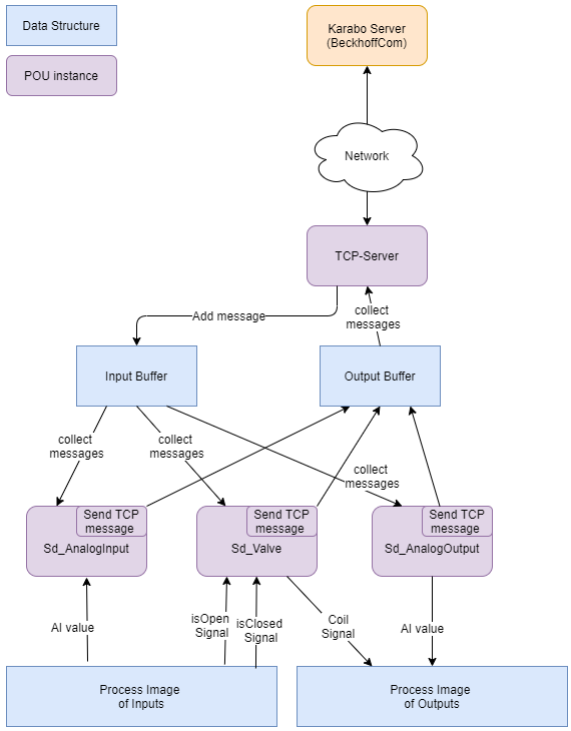


Figure 1: Structure of the Current Library.

pleteness. Firstly, it would be helpful for a textual description of each command and property to also be sent to the control system. This is currently only available in the online documentation of the library and not linked to the control system where it is used. Additionally, a description of the error states should be added provide context in comparison to the existing error code. These additional features could overcome the need for manual development of the corresponding devices on the control system, known as "Karabo devices".

Partial Regeneration and Handling of Custom Code in Projects The process of building a PLC project is automated to a high degree at EuXFEL, but the way it is currently implemented requires a complete re-build of the entire solution, overwriting any code that had been previously added. Despite the feature-richness of the tools available, this complete regeneration process has caused many issues when only a minor change is required. Due to these limitations, the future library and associated tools need to be more flexible, providing the ability to generate one or a set of sections of the PLC project, or a particular feature set.

Additional Abstraction of the Library The current version of the library does not utilize abstraction layers in the architecture, and so all softdevices have to deal with all the various functional layers that exist in the library. While developing a softdevice, the developer has to bear in mind very low level details of the terminals that might be connected to the softdevices while at the same time considering high level aspects, such as how to structure the interface for the end user, who interacts with the softdevice through the control system. To faciliate understanding and implementation of softdevices, it is planned to add a layered architecture to the future library. This should reduce the complexity of softdevices and thereby reducing the likelihood that bugs are introduced, while producing testable code. A useful feature when tackling new softdevices. In addition to this, it is foreseen that a higher layer to cover additional automation of systems and processes e.g. to handle beamline components may be desired. See Device Abstraction Layer.

Relinking on Runtime There have been instances when during operational periods, hardware (EtherCAT terminals/channels) had to be reallocated. Within the current library, hardware I/O are directly linked to the softdevice. This is a severe restriction due to the fact that most of the links can only be relinked with a rebuild and redeploy of the TwinCat configuration - something which is not possible during operation of the system. The future library will overcome this limitation through the additional TcHAL and an interface manager. All EtherCAT terminals will be abstracted by a dedicated call per terminal type, where all terminals and channels will also have a defined interface. These interfaces will on runtime be registered with the "interface manager" Programming Organisation Unit (POU), implemented as a singleton within the TcHAL, which, is responsible for handling both the interfaces to the TcHAL and the propagation of information to any compatible softdevice.

Generate Terminal I/O Linking Map Extending on from the ability to relink hardware during run-time, the information related to the current linking should be obtainable such that any previous wiring diagrams can be amended to reflect reality. Currently, there is no way to keep track or compare what has been configured, to what is expected. Not only does this hamper debugging, but it also adds confusion and produces unreliable documentation. Further utilisation of the configuration service tool will enable the exporting of this information.

Simplified Peer-to-Peer Communication Functionality for communication between softdevices already exists, however it does so in a rather complicated manner and involves a deep understanding of the device implementation. Softdevices are required to send "pairs" to other devices to issue commands or read and write requests, more or less utilising the same interface methods as the control system. This is not developer friendly or easily human-readable. An alternative method is to use a globally defined shared memory, which enables softdevices to read a predefined set of values for any softdevice on the system, again, requiring in depth knowledge as it is up to the developer of a softdevice to decide which properties are put into this shared memory. As this information is not self describing, this is a fragile way to exchange information, requiring type casting of each property, while considering that a data type might change over time.

Testability of the Library The structure of the current library makes it very hard to test individual aspects, functions and components, which in turn makes it practically

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR013

impossible to unittest. This leads to a situation where a significant amount of manual testing is required, which is time-consuming and highly error-prone. With the proposed TcHAL, softdevices will become largely decoupled from the hardware and also from the SCADA system communication via the means of dedicated wrapper classes. As such, softdevices can be tested within each interface, and interaction between each level of abstraction can be completed with its own test case, opening up a better way to perform testing.

Resolve the Interlock Feature of the Current Library The current library implements a feature called "interlock", which allows a device to react in defined ways upon a set of predefined conditions. The interlock feature is intended to be used for equipment protection, however, over time has been extended to perform automation on the PLC as well, which is needed where a specific reaction time has to be achieved. The implementation of interfaces as well as layers within the future library will allow us to overcome the rigidity of the feature as currently implemented. The new structure will enable peer-to-peer interaction in a much more diverse and safer fashion. Combined with Simplified Peer-to-Peer Communication, it would be possible to write devices where the "interlock" feature becomes redundant.

Simplified Device Development Due to the implementation of layers and interfaces that are tailored to the needs of the PLC device developer, developing devices will become simpler and more efficient. This will open up the ability for "beamline component" experts to write their own devices, initially within the projects, then, eventually implemented within the "beamline component layer" of the library. Needless to say, the transformation of specific devices implemented by a "beamline component" expert within a project into a generic device becoming part of the future library, will be a joint effort of the "beamline component" expert and a PLC library developer.

Self-contained PLC System In order to make the PLC systems at EuXFEL self-contained, it is necessary to implement a solution to store and restore configuration data of PLCs on the Industrial PC (IPC) that is running the PLC. To be able to restore the previous configuration state of the PLC after a restart as well as after an update.

Structure of the Library

The structure of the future library will consist of at least two different layers; one for the hardware (field bus terminals) abstraction, provided by TcHAL, and one for the softdevice implementation, otherwise known as the Device Abstraction Layer (DAL). Communication between components across different layers will be carried out through interfaces, a concept within the IEC 61131-3 3rd edition which was not yet available when development of the initial library began within the TwinCAT 2 environment. The structure of the current library can be seen in Fig. 2.

Hardware Abstraction Layer The Hardware Abstraction Layer (HAL) provided by the library TcHAL is intended to abstract the EtherCAT fieldbus terminals. The TcHAL provides POUs for each fieldbus terminal used at EuXFEL. It will configure the terminal according to its configuration, provided through a configuration interface of the terminal-POU. Data will be periodically exchanged with the terminal via the EtherCAT master and translated into meaningful units, preferably in International System of Units (SI) where possible. Other components of the system will then obtain these values as required. These interfaces will be tailored to the needs of components of the DAL, see Device Abstraction Layer. As a result, the HAL aims to provide a clean and easy to interface into the hardware for the DAL device developers.

Device Abstraction Layer The DAL which sites above the HAL, will be the second layer of the future library as seen in Fig. 2. It is on this layer that softdevices are to be implemented, without the complexity of any hardware configuration.

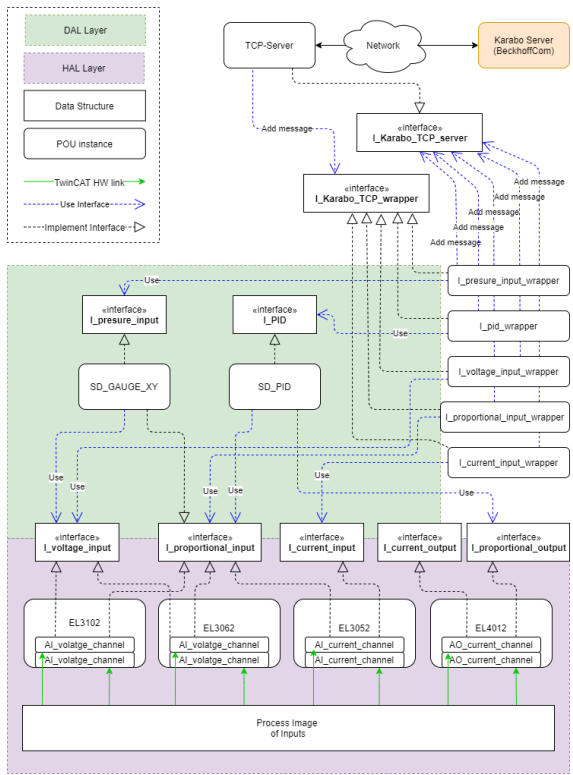


Figure 2: Structure of the Future Library.

Communication Protocol

The future library will implement an extended version of the custom TCP-protocol of the current library. This is predominately required to overcome the limitations seen during operation, specially with large PLC-projects which encompass hundreds of devices.

Extended Self-Description The current implementation of the self-description sends a list of all softdevices on connect. It also sends a description of each softdevice-class including all properties and commands, with the name,

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR013

data type, access level, unit and unit prefix of the associated properties and commands on a device-instance or object level, and also for any type variations which deviate from the standard device-instance. The future library will extend this self-description with a description of each command and property alongside the name, aimed at providing additional information to the end user. Additionally, a mapping between available commands and properties for a given Karabo state is planned. Beyond the existing message types, an additional message type to define error codes with their associated description is a future feature.

Separation Between Self-Description and Operation The current version of the library implements the self-description of the system on the same server (port) as operationally related communications. This is not ideal, since the self-description is sent to the client on connect, not on request. When this occurs, it creates a notably large load on the client using a protocol which is not optimized for large amounts of data, but rather for small amounts of data at high data rates. It is planned to differentiate the data required for operation: device state updates, reads and writes, and command requests; and the self-description: description of all devices on the system and associated details such as which commands and properties are available. The new self-description data will be provided on a dedicated port, with JavaScript Object Notation (JSON) being the intended format. This fits in nicely with the other planned changes given its amenability strings.

Additional Tooling

The current PLC library uses a set of tools which where predominantly implemented in Python, and one in C#. There one commonality is that they are not part of the development environment of the PLC developers. Some tools are part of continuous integration (CI)-pipelines, and others are to be used from the command line on dedicated development machines. Due to the fact that these tools were not envisioned at the beginning of the current library development, they rely on unconventional ways to retrieve the necessary information from the library. Much of this information is gathered through several iterations of source code parsing, which makes the "interface" quite fragile. Given that the code and/or comments written are not checked by the PLC compiler (IDE) or any subsequent build tool to ensure adherence and compatibility, this process becomes highly susceptible to error. This is exacerbated by the fact that errors may not be caught until after the various CI pipelines have run, or only during PLC project builds, making the entire process tedious. In order to provide a more convenient and comprehensive solution for PLC developers, it is aimed to have these tools integrated into the PLC IDE via plugins to Visual Studio. In conjunction with the other changes planned for the library, the parsing of source files will become redundant. Ultimately, this means the tools will be re-written in a .Net language as an overarching "Configuration Manager" as well as a "Development Toolbox".

Check and Configure Connected Hardware The "Configuration Manager" will be implemented as a tool to check the configured hardware of a TwinCAT project. It will retrieve all the necessary information with how to set up each terminal of the project, based on the requirements of the TcHAL POUs in the selected EuXFEL PLC library. The settings are to be applied to the terminals, skipping over and flagging those that are unsupported or incompatible.

Generate and Link TcHAL POUs In order to generate and link all TcHAL POUs according to the hardware configured in a TwinCAT project, the "Configuration Manager" will contain a tool that provides the user with an easy-to-use interface.

Apply EuXFEL Terminal Naming Convention Currently, there are two different naming conventions at EuXFEL defining how to name automation components such as fieldbus terminals and connected equipment. These naming conventions are reflected within the hardware configuration of TwinCAT projects and in the hardware abstraction. The "Configuration Manager" plugin will contain a tool, that will allow PLC experts to apply one of these naming conventions to hardware configurations of a TwinCAT project comfortably. This will reduce the work while using scanned hardware, in this case TwinCAT applies generic names to all scanned components, or switching from one to the other naming convention.

Generate Device Skeleton To provide consistency across the future library and projects with custom devices, a tool to support device development is planned as a part of the "Development Toolbox" plugin. It will allow the user to generate a device skeleton on any layer (see Additional Abstraction of the Library) of the future library (or project). It will give the developer the opportunity to define which interface(s) should be implemented and which name the device should have. It will then generate skeleton code for the POU adhering to the EuXFEL PLC library naming convention, as well as boilerplate code to implement methods and commands.

Generate Karabo Wrapper Class This tool which is part of the "Development Toolbox" plugin, is to provide the user with a POU template generator, which can be pointed to an interface and is made available to the SCADA system in order to ease implementation of the interface wrapper class.

EuXFEL PLC Linter Expanding upon the naming convention tool, an additional feature will be a check function which can be performed across the entire code base. It is intended to be added to the "Development Toolbox" plugin.

Architecture Violation Checker In the long run, a feature to define architectural rules will also be added to the "Development Toolbox" plugin. Additionally, a checking feature will be provided to ensure the code adheres to the defined rules.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR013

Configuration Management

The current library does not have a proper concept to handle persistent configuration data of softdevices. It is envisioned to implement a dedicated configuration interface on every POU that is required to handle persistent data. On the IPC running the PLC, a service will be implemented using Beckhoff's ADS-protocol to retrieve and restore data as requested by the POU. The data will be stored on the IPC with the possibility of an additional backup in the event that the IPC crashes. This approach will not only allow the system to retrieve configurations after a restart or a power cut of a PLC, but also after an update or partial reconfiguration of a PLC.

MIGRATION STRATEGY

It is intended to perform the refactoring of the current library in four major steps while ensuring as much backwards compatibility with the currently running projects, build tools and finally, control system, as is feasible. If a new feature breaks compatibility, as is the case with the evolved communication protocol (between the PLC and the control-system), a migration strategy has to be formulated, to ensure adequate uptime across all involved systems.

Hardware Abstraction Layer

The hardware abstraction layer will be added as soon as all the terminals used at EuXFEL have been implemented in the new library. As the existing softdevices are linked directly to the hardware, all softdevices will have to be migrated one by one to adapt to the new structure and interfaces provided by terminal POUs of the TcHAL. To aid this process, a compatibility adapter will be made available for the existing devices, linking them to the new interfaces on the other side. These adapters will be removed once all softdevices have been migrated to incorporate in TcHAL interfaces.

Layers and Interfaces

In order to establish a layered architecture in the future library, it is necessary to refactor all softdevices to implement commands and properties via methods organized in interfaces. The refactoring can be completed softdevice by softdevice, ensuring no unintended effects on the overall system during the migration period. This will make it possible to add the next layer onto the DAL. The TcHAL Hardware Abstraction Layer will make up the second layer.

Updated Communication Protocol

Lastly, migrating to a new communication protocol is planned once the design has been completed and a client is available. As soon as the self-description service, which is serving the JSON self-description is done, the existing implementation of all softdevices will be changed to not send a self-description on connect anymore. The structure of the future library enforces a decoupling between the softdevice and communication component to the control system. All new softdevices will be implemented following this approach, whereas the old devices will get adapters until they are migrated internally.

Add Configuration Persistence

Softdevices that need to store/restore configuration parameters will make use of configuration interface they define, being implemented by configuration-devices. These interfaces will be passed into the softdevice via a reference, which will allow the softdevice to figure out if the interface is implemented by a configuration-devices on runtime. If this is not the case, the softdevices can make use of the old hardcoded configuration values, which are passed via inputs. This approach will make a piecemeal migration possible, softdevice by softdevice.

SUMMARY

Through several years of refinement, it can be seen that the existing PLC Library requires a significant restructuring. It is hoped that though refined abstraction layers and advanced tooling, we can achieve an easily (re)configurable PLC library which can provide further automation and efficient development and deployment.

REFERENCES

[1] T. Tschentscher *et al.*, "Photon beam transport and scientific instruments at the European XFEL," *Appl. Sci.*, vol. 7, no. 6, p. 592, 2017. doi:10.3390/app7060592

[2] S. Hauf *et al.*, "The Karabo distributed control system," *J. Synchrotron Radiat.*, vol. 26, no. 5, pp. 1448–1461, 2019. doi:10.1107/S1600577519006696

[3] S. Huynh *et al.*, "Automatic Generation of PLC Projects Using Standardized Components and Data Models," in *Proc. ICALEPCS'19*, New York, NY, USA, 2020, pp. 1532–1537. doi:10.18429/JACoW-ICALEPCS2019-THAPP01

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR021

EPICS IOC AND PVs INFORMATION MANAGEMENT SYSTEM FOR SHINE

Huihui Lv, Yingbing Yan†, Qingru Mi, Guanghua Chen
Shanghai Advanced Research Institute, Chinese Academy of Sciences, Shanghai, P.R. China

Abstract

For Shanghai HIgh repetition rate XFEL aNd Extreme light facility (SHINE), EPICS is adopted as standard to build the complex control system which involves hundreds of IOCs and millions of records. Most of IOCs run in industrial control computers as soft IOC. However, with the large amount of PVs, the need has emerged to develop remote tools to monitor all the channels and IOCs. One application backed by MySQL is designed, running periodically to interact with EPICS system where to take data from run-time databases via Channel Access and pvAccess. We embed scripting codes into the IOC startup script to realize that as soon as the IOC starts up, the information of the server's address, the IOC installation path, and all the records maintained by the IOC will be automatically pushed to the application and then stored in the database. With this necessary information, we could access all the active IOCs and PVs. Another web application is designed to render servers, IOCs and PVs data in MySQL on the web to give us an overall running status of the control system. We also fully consider the modularity and portability for the applications to apply and extend in none-SHINE environments.

INTRODUCTION

Motivated by the successful operation of X-ray FEL facilities worldwide and the great breakthroughs in atomic, molecular, and optical physics, condensed matter physics, matter in extreme conditions, chemistry and biology, the first hard X-ray FEL light source in China, the so called Shanghai HIgh repetition rate XFEL aNd Extreme light facility (SHINE), is under construction. SHINE will utilize a photocathode electron gun combined with the superconducting Linac to produce 8 GeV FEL quality electron beams up to 1 MHz repetition rate [1]. Hundreds of IOCs, located in Shaft #1, Shaft #2 and Shaft #3, will directly or indirectly control almost every equipment distributed in tunnels of the injector, the superconducting linac, and three FEL undulator lines, with the total length of approximately 3.1 kilometers long(as shown in Figure 1). Shaft #1 is a building at the end of the injector, Shaft #2 is at the end of linac, and Shaft #3 is at the end of undulator lines.

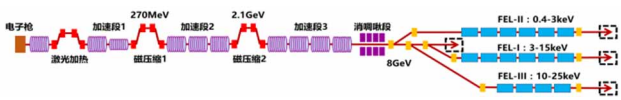


Figure 1: SHINE Accelerator Layout.

The control system involves heterogeneous equipment and interfaces with a number of different subsystems. Several IOCs will be deployed in one server, in which hundreds of PVs perform real-world I/O and local control tasks. But we suffer from a lack of high level tools to centrally manage all the IOCs and PVs and help us to verify which PVs are running on a specified IOC, and how many IOCs are running on a specific server. For an IOC, we also need to know which PVs are active. In this case, we develop the application to obtain all the PVs from EPICS system and store them in the MySQL database. In order to search through data easily, we also develop the web application to represent the information of the database. Details are described in the following sections.

ARCHITECTURE OVERVIEW

The data flow view of the architecture as shown in Figure 2 can be divided into four functional modules, namely database, import service, PV service and web service. The database is a general storage container for the properties and real-time values of servers, IOCs, EPICS records and PVs. The import service aims to import the initial information such as record types, field name lists and field types into the relational database from a spreadsheet. PV service fetches PVs from EPICS run-time databases via Channel Access and pvAccess. At the same time, it retrieves the running information of servers such as the available free memory, CPU load, etc. Then it saves all the data into the database. Web service is mainly used to render the data on the web to give us an overall running status of EPICS control system for SHINE.

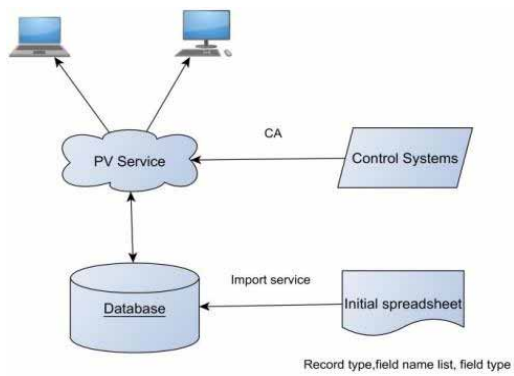


Figure 2: System Architecture.

Database Schema

The database saves the properties as Name/Value pairs. Details are illustrated in Figure 3. More specifically, it is needed to store the information of server, IOC, record and PV. For the server, the database is not only to store static attributes, like environment variables related to EPICS, but also to store the runtime information, for instance, when the server is started, when the server is scanned, CPU load,

yanyingbing@zjlab.org.cn

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR021

etc. These are handled by tables of server, server_env, epics_env, and server_runtime, respectively. Generally speaking, there are several environment variables set up for the server to run EPICS system, such as EPICS_BASE, EPICS_EXTENSIONS, EPICS_HOST_ARCH, EPICS_CA_AUTO_ADDR_LIST,EPICS_CA_ADDR_LIST and so on. They are stored as Name/Value pairs as shown in table server_env. Similarly, others environment variables, such as EPICS_TIMEZONE, EPICS_CA_CLNT_CNT, EPICS_CA_CONN_TMO, EPICS_TS_NTP_INET, EPICS_VERSION, stored in epics_env table are also designed as Name/Value pairs. The running information, like available free memory, CPU load, buffers are archived with timestamp when the server is scanned.

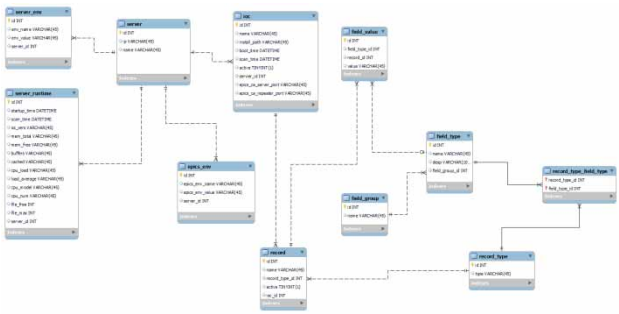


Figure 3: Database Schema.

The ioc table is to store the installation directory, startup time, active/inactive state and the UDP port to which an IOC is listening (5064, 5066...). What's more, a foreign key is inserted into the table that represents the server for One-to-Many relationship between them. The other six tables are used to store records and PVs. However, their relationship is a little complicated. A diversity of data from hardware are obtained by IOC as different record types, such as ai, bi, mbbi, mbbo and so on. Moreover, each record contains dozens of fields for different purposes, for example, INP field used to obtain the record's input, EGU and DESC used to present meaningful data to the operator. These data are stored in tables of record_type and field_type. Similarly, one field will be used in many record types. For instance, the DESC filed is utilized in multiple records. A many-to-many relationship occurs here, which is handled by a joining table that sits between the two tables. Its purpose is to store a record for each of the combinations of these two tables. It might seem like a bit of work to create, but it provides a much better data structure. A specific record which belongs to a typical record type involves a fixed set of field types. The values of fields, the so-called PVs are stored in field_value table. Field type/value is designed as Name/Value pairs for storing, that could simplify the overall complexity of the database schema.

Import Service

We create a spreadsheet with custom CSV layout to store the field types included by each type of record, that allows for easy import into the database. The tabs of ai and bi are shown in Figure 4. A single tab is initialed for each type of record. There are 15 record types we use frequently for SHINE, called ai, ao, bi, bo, longin, longout, mbbi, mbbo, stringin, stringout, waveform, subArray, compress, calc, calcout. Types vary from one facility to another.

Name	Description	group
VAL	Value Field	Read Write and Convert
INP		
DTYP		
LINR		
RVAL		
ROFF		
EGUF		
EGUL		
AOFF		
ASLO		
. . .		

Name	Description	group
OMSL	Output Mode Select	Desired Output
DOL	Desired Output Location (an Input Link)	Desired Output
OIF	Out Full or Incremental	Desired Output
VAL	Desired Output	Desired Output
OUT	Convert and Write	Read Write and Convert
DTYP	Device Type	Read Write and Convert
LINR	Type of Conversion	Read Write and Convert
RVAL	Raw Value	Read Write and Convert
ROFF	Raw Value Offset	Read Write and Convert
EGUF	Engineering Units Full	Read Write and Convert
. . .	. . .	. . .

Figure 4: CSV layout.

Import Service aims to import the data in the spreadsheet, as well as their relationships into MySQL database at one time. Apache POI [2] helps us to easily read and manipulate the excel files in our application by writing the Java code. Of course, duplicate records in the spreadsheet will be checked and consolidated into single records in the database.

PV Service

PV Service accesses EPICS Channel Access channels using ca library to get PVs asynchronously. Ca is a pure Java Channel Access client implementation developed by PSI [3]. As shown in Figure 5, PV Service is responsible to store PVs into the database as well. A channel is established for each of several fields involved in a record, and a channel group is formed by multiple channels, which communicates with IOC asynchronously. Meanwhile, SSH protocol is used to connect to IOC servers to execute shell commands for getting running status of the IOC server, such as CPU load and memory usage. JSch [4], as a pure Java implementation of SSH2, allows us to integrate these functions into our Java programs.

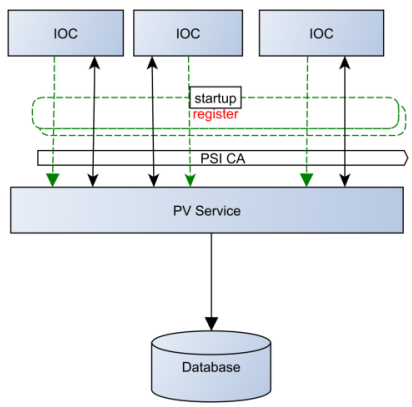


Figure 5: PV Service Architecture.

To be specific, the operation process of PV Service is divided into three stages: register, scanning and updating. First of all, necessary scripts (Figure 6) are embedded into the startup shell (st.cmd) in order to obtain the information of the IOC, such as the installation path, the startup time, the record lists, and so on. All the information is automatically written to a text file, which is then delivered to the

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR021

server running PV Service, making use of the combination technologies of rsync+ssh+lsync.

Second, the program parses the text file for the information of the installation path, IP address, epics ca server port, epics ca repeater port and lists of the record for an IOC. With the IP address, we can access the server to get the memory usage information: mem total, mem free, buffers and cached, the CPU usage information: CPU model, CPU number, as well as the information of the file descriptors: file free, file max. The IOC related information is stored to ioc table, while the record related information is inserted into record table.

```
epicsEnvSet("PVINFOFILE", "/home/iocusr/pvinfo/${IOCNAME}.pvinfo")

system "echo IOCName:${IOCNAME} > ${PVINFOFILE} "
system "echo path:${TOP} >>    ${PVINFOFILE}"
system "echo -n Date: >> ${PVINFOFILE}"
date >> ${PVINFOFILE}

system "echo env_params: >>${PVINFOFILE}"
epicsPrtEnvParams >> ${PVINFOFILE}

system "echo record:    >>    ${PVINFOFILE}"
dbl >> ${PVINFOFILE}
```

Figure 6: Part of Scripts.

Finally, the Java scheduled task is set up to obtain PVs for all the fields in a record via CA protocol and store the data in the database. Therefore, the data are always in the latest state.

Web Service

The web service is built with JavaServer Faces [5] technology, together with PrimeFaces UI framework. It is deployed in Glassfish container and serves pages viewed in the web browser hierarchically. There are three panels illustrated in Figures 7, 8 and 9 to display the information of the server, ioc and record, respectively. When the web page appears, it shows a simple drop-down menu that allows you to select a server on the left side. Then the relevant information will be demonstrated on the right side, coupled with all the IOCs deployed in the server. The server's information includes the IP address, the startup time, OS version, CPU model, CPU cores, the available memory size, the unused memory size, CPU load, etc.

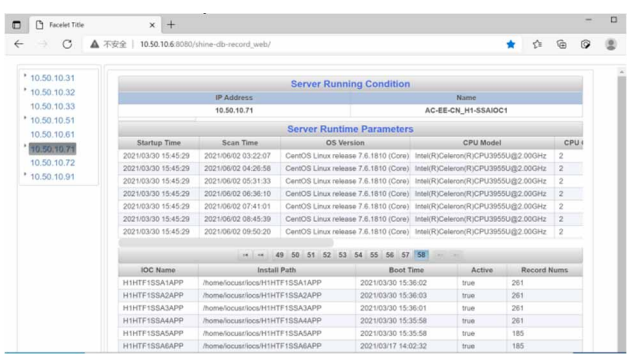


Figure 7: Display interface of the server.

The IOC panel as shown in Figure 8 will display not only the running information of the IOC, but also all the records included in the IOC. The information of the IOC includes the installation path, the startup time, epics ca server port, epics ca repeater port, running status, the number of connected and unconnected records. The record panel shown in Figure 9 displays all the fields' information.

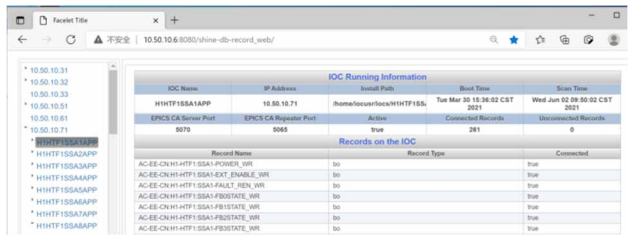


Figure 8: Display interface of the IOC.

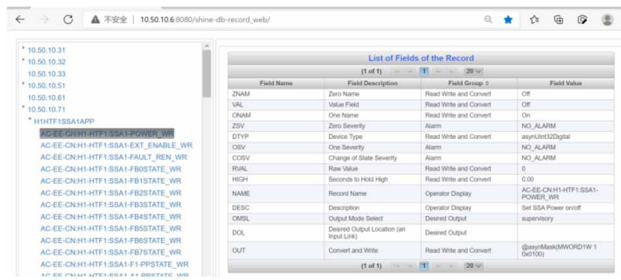


Figure 9: Display interface of the record.

CONCLUSION

We succeed in developing EPICS IOC and PVs information management system for SHINE. MySQL database is utilized to store all the information, of which the running data from the control system is fetched via PSI CA interfaces and the running states of servers are obtained via SSH protocol. The web service allows web pages to render for the information in MySQL database. The system has been operating stably for more than one year in the cryomodule test facility for SHINE [6]. With this system, we can easily and fully grasp the holistic control system operation.

REFERENCES

[1] Kai Li and Haixiao Deng, "Systematic design and three-dimensional simulation of X-ray FEL oscillator for Shanghai coherent light facility", *Nucl. Instrum. Methods*, vol. 895, pp. 40–47, 2018. doi:10.1016/j.nima.2018.03.072

[2] Apache POI, https://poi.apache.org/

[3] CA Library, https://github.com/channelaccess/ca

[4] JSch, http://www.jcraft.com/jsch/

[5] JSF, https://www.oracle.com/java/technologies/javaserverfaces.html

[6] H. Y. Wang, G. H. Chen, J. F. Chen, *et al.*, " Control System of Cryomodule Test Facilities for SHINE", in *Proc. ICALEPCS'21*, Shanghai, China, Oct. 2021, pp. 353-357. doi:10.18429/JACoW-ICALEPCS2021-TUBR04

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR022

DATA ACQUISITION SOFTWARE PIPELINE FOR THE COMMISSIONING OF THE LoKI SMALL ANGLE NEUTRON SCATTERING INSTRUMENT

J. Walker*, S. Alcock, M.J. Christensen, J.E. Houston, K. Muric, W. Potrzebowski, T.S. Richter
European Spallation Source ERIC, Lund, Sweden
Davide Raspino, UKRI-STFC, ISIS Neutron and Muon Source,
Harwell Science and Innovation Campus, Chilton, Oxfordshire, UK

Abstract

The LoKI Small-Angle Neutron Scattering (SANS) instrument will be one of the first instruments to be commissioned at the European Spallation Source (ESS), and will contribute to the early science programme. The detector for the instrument was tested at the ISIS neutron source facility in March of 2022, and this paper outlines the data acquisition software pipeline. It consists of a readout master, an Event Formation Unit (EFU), an instance of Kafka, a NeXus file writer, and, the data reduction software, Scipp. The readout master is responsible for synchronising detector readout timestamps with an external reference, and aggregating those readouts into UDP packets sent to the EFU. The EFU processes the readout data to produce event messages containing a location and time of arrival for each neutron event detected, and acts as a Kafka producer sending event messages to Kafka. The NeXus file writer consists of a Kafka consumer that compiles event messages and other experiment data from a given time interval into a single NeXus file for further analysis. Finally, Scipp is a data reduction python library used to visualise and analyse the experiment data after an experiment is completed.

INTRODUCTION

The European Spallation Source (ESS), co-hosted by Sweden and Denmark, is currently under construction in Lund (Sweden), with the first neutrons expected in 2024. A number of detectors are being developed for the facility, one of which is for the LoKI Small-Angle Neutron Scattering instrument [1, 2], which is being developed in colaboration with Science and Technology Facilities Council (STFC) in the U.K. LoKI will make use of ^{10}Boron-Coated Straws (BCS) from Proportional Technologies Inc. (USA). In order to obtain acceptable efficiencies, these detectors consist of 7 boron-coated copper straws packed within 1 inch aluminium tubes, with each copper straw wired as a position sensitive detector. On LoKI, these 1-shell-escape inch aluminium detector tubes will then be packed into arrays to make detector panels, which will be placed in 4-panel banks around the beam at ~1.3 m and ~4 m from the sample, and a single panel bank on a carriage, which will move between ~5 m and 10 m from the sample positions. Given the number of detector straws in this design, in order to minimise detector signal cables, the multiplexing method will be used. The multiplexing method consists of a resistive chain between the 7 straws at each end of the aluminium tube, resulting in four signals from each each tube rather than 14. When a neutron is detected, 4 amplitude values are then produced, from which the location of the neutron event can be calculated. A smaller subset of this detector consisting of 128 tubes was used for the tests outlined in this paper. A diagram of this detector is shown in Fig. 1, showing the layout of the tubes, and the layout of the straws within each of those tubes is shown in Fig. 2.

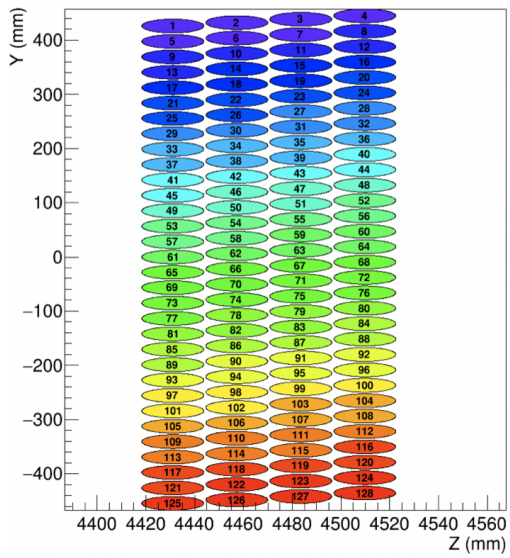


Figure 1: A diagram depicting the tubes of the subset of the LoKI detector tested at ISIS in March 2022.

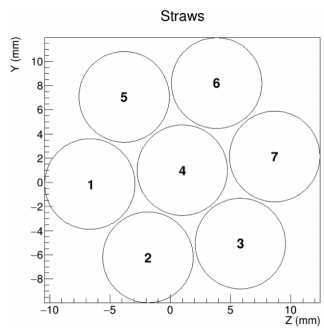


Figure 2: A diagram depicting the straws within each tube of the LoKI detector.

The detector was tested on the LARMOR instrument at the ISIS facility with the full data acquisition pipeline [3]. This pipeline will be used in production at the ESS facility. It consists of the following key components, developed by

* jennifer.walker@ess.eu

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR022

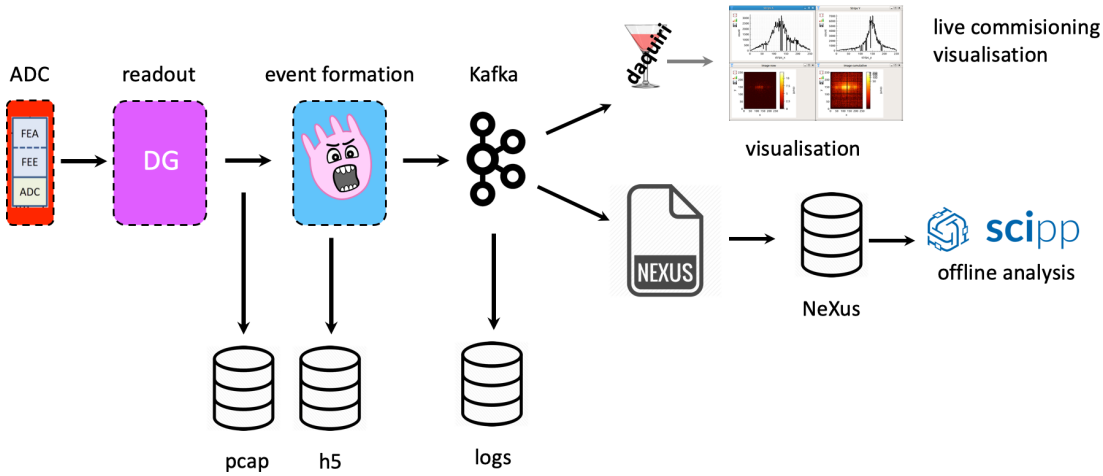


Figure 3: The data acquisition pipeline used at the LoKI test at ISIS in March 2022.

ESS and in-kind partners, each of which will be outlined in this paper.

- Readout Master
- Event Formation Unit
- NeXus File Writing
- Scipp Data Reduction

The data acquisition software chain is shown in Fig. 3, from the Analogue-to-Digital Converter (ADC) readouts on the detector, through the Readout Master [4], to the Event Formation Unit [5], where unprocessed data for this test is also stored in the form of pcap and h5 files for later analysis and reprocessing. The pipeline then goes on to the Kafka message broker[6], and services that consume data from Kafka in this case include DaqLite [7] and the NeXus file writer. DaqLite is a live data visualisation tool that allows for monitoring of the detector activity early in the acquisition chain. It displays the cumulative hit count at each voxel of the detector. The files written by the NeXus file writer are then ready for data reduction in Scipp [8, 9].

READOUT MASTER

Detector systems convert neutron events into electrical charges, which can be amplified, shaped, and digitised. Large detector systems may comprise many thousands of such outputs. The digitisation may be performed by a series of discrete ADCs, or by more sophisticated multi-channel ASICs. These devices must be read out, typically by one or many FPGAs. The function of the ESS Readout Master is to simplify the communication between these FPGAs and the relevant downstream systems provided by the Integrated Controls Group (ICS) - slow control and timing - and the Data Management and Software Centre (DMSC) - event formation. The Readout Master essentially acts as a point of aggregation, allowing for a common, standardised interface between the detector readout and the ICS/DMSC.

For the ICS interface, the Readout Master must handle slow control and timing distribution. For the timing interface, the Readout Master instantiates the open-source MRF (Micro Research Finland) EVR (Event Receiver) firmware, allowing it to connect to the ESS facility MRF timing system. This allows for the recovery of the ESS facility clock, timestamp and other useful events distributed by the MRF system, for example those related to the accelerator pulse. The clock and timestamp are forwarded to so-called "Front End Nodes" (FENs) which interface to the front end FPGA/ADC/detector chain. The connectivity between the Readout Master and the FENs is achieved through 8B/10B encoded MGT-driven (Multi-Gigabit Transceiver) optical fibres links arranged in a ring topology. Up to 12 rings of 31 FENs are supported, allowing for a scalable system from 1 to 372 FENs, with each FEN typically supporting a minimum of 64 ADC channels. Each FEN can recover the ESS clock from the MGT links. The timestamps are synchronised for each FEN using a simple round-trip delay calculation. Timestamp distribution is accurate to within 100 nanoseconds, and is guaranteed not to drift with respect to the ESS facility timestamp.

For the slow control interface, the Readout Master instantiates a light-weight UDP stack, which communicates with the EPICS IOC (Input Output Controller) using a standard Gigabit Ethernet link. The slow control system is used to read and write to all registers within an instrument readout system. The EPICS IOC acts as the slow control Master, and the Readout Master is responsible for routing register requests to the relevant address space. If the required register is on a FEN, the request and response are handled by the same optical links which carry the timing information.

The FENs package neutron event data and send it back to the Readout Master over the same optical links that carry the timing information. The Readout Master aggregates these into UDP packets, appends relevant metadata, and then forwards them to the EFU (Event Formation Unit) as jumbo frames over one or two 100 Gigabit Ethernet links [10].

EVENT FORMATION UNIT

The Event Formation Unit, a Linux server application written in C++, receives data from the Readout Master from the previously described 100 Gigabit Ethernet links. The data format output by the Readout Master for consumption

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR022

by the Event Formation Unit consists of the ESS data header, which among other things defines the pulse time and previous pulse time, and a series of LoKI readouts. The format is shown in Fig. 4. Each of these readouts represents a single neutron event, and all readouts in a single packet belong to the same pulse time, indicated in the ESS header.

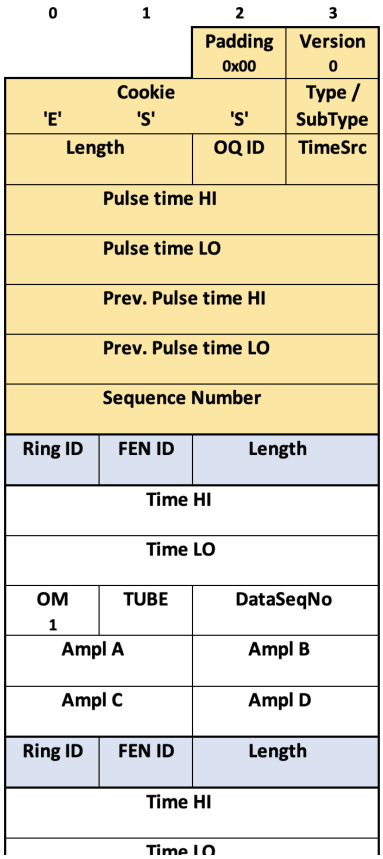


Figure 4: The ESS Header data format, in yellow, followed by a series of LoKI readouts. The format is displayed as 4 bytes wide and offset by 2 bytes in the diagram for legibility.

The Event Formation Unit is responsible for processing these readouts and outputting Kafka messages containing their pulse time, time of flight, and pixel ID. The pixel ID is an integer value that represents the location on the detector where each neutron is detected. The definition of these pixel IDs is referred to as the Logical Geometry of the detector, and is shown in Fig. 5. Mapping from the digital identifiers in a readout - the Ring ID, FEN ID, and Tube number - is defined in the Digital Mappings, shown in Fig. 6.

The straw number and the location along each straw are calculated from the four amplitude values per readout according to the following equations:

$$LocalStraw = \frac{B+D}{A+B+C+D}6$$

$$Position = \frac{A+B}{A+B+C+D}(2^N - 1)$$

With the resolution of position, the value N, being 512 for these tests.

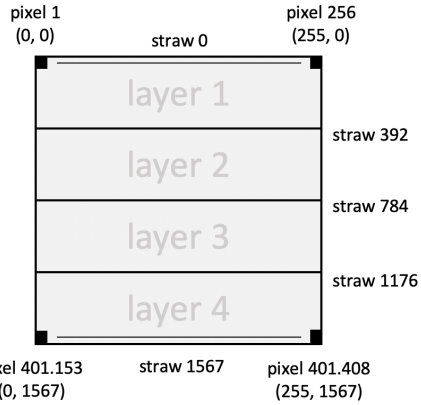


Figure 5: The Logical Geometry, defining an integer value per position on the detector.

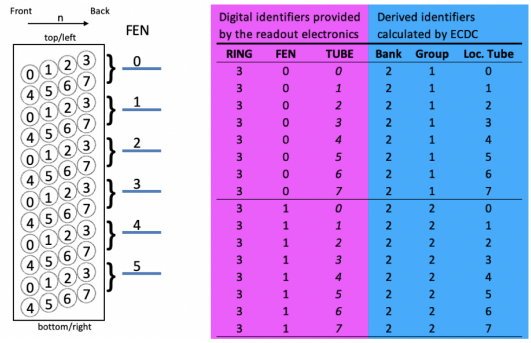


Figure 6: The Digital Mapping, defining what Ring ID, FEN ID, and Tube numbers map to specific tubes in the detector.

From the local straw value, i.e. the straw number within a tube, and the global tube position, a global straw number is calculated. This then allows the final pixel ID calculation:

$$PixelID = 1 + Position + GlobalStraw * 2^N$$

Finally, the Event Formation Unit produces Kafka messages using the ev42 flatbuffers schema consisting of the following fields:

Field Name	Type	Description
source_name	string	Field identifying producer, commonly detector name
message_id	ulong	Consecutive numbers, used to check order of messages
pulse_time	ulong	Nanoseconds since Unix epoch of given pulse time
time_of_flight	uint	Nanoseconds since pulse for given event
detector_id	uint	Pixel ID, identifying where neutron was detected

NeXus FILE WRITING

Data file writing is performed using an in-house developed file writer, called Kafka-to-nexus, written in C++. As the name indicates, it retrieves all the collected data during the experiment from a Kafka cluster and writes it to a NeXus

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR022

```
{
    "module":"ev42",
    "config":
        {
            "topic":"loki_detector",
            "source":"loki"
        }
}
```

Figure 7: The LoKI event data configuration for the file writer. As can be seen, the ev42 flat buffer schema is used to serialise the event data. The file writer can fetch the collected data from the loki_detector Kafka topic.

file. NeXus is a format based on HDF5 that is commonly used in muon, neutron and x-ray science. Attributes are used extensively; for example, group attributes are used to define the NeXus class of a group written to the file.

At ESS, the overall structure of the NeXus file and the type of data written is defined by a template JSON file, which is provided to the file writer in a start job message sent to a file writer command Kafka topic. In addition to where and how to fetch event and measurement data, the template JSON file also contains static data, which can be written directly by the file writer from the NeXus template structure. One such instance of static data would be any beforehand known experiment information, such as experiment title or beam users. Any dynamic data that needs to be written in the NeXus file is described by the template file where such a stream data configuration, for event data using the ev42 schema, is shown in Fig. 7.

The pixel geometry information and ID mapping is also provided through the NeXus template file. As the shapes are uniform for each pixel in the LoKI detector, the NXcylindrical_geometry class according to the NeXus standard is used to describe a pixel. Data sets x_offsets, y_offsets and z_offsets are used to provide each individual pixel with Cartesian coordinates with the origin at the sample position. Transformations according to the NXtransformation class are used to place the detector correctly in the global coordinate system relative to the sample (origin). Both the data sets and the pixel shape groups are items in an NXdetector group. Additionally, in the NXdetector group there is a data set called detector_number that defines the mapping of pixels to pixel IDs. This mapping is defined by the Logical Geometry of the detector in combination with physical dimensions specified in technical specifications, such as CAD drawings. Event data with the configuration according to the one in Fig. 7 is written to an NXevent_data class, which is a subgroup of the NXdetector group.

A start job command containing the start time of the write job and the previously described JSON template file is sent to the file writer command topic. A file writer is selected from a pool of currently idle and available writers, and it will start writing the data based on the provided template. First of all, the NeXus file writer defines the overall structure of the file and populates it with the static data such as pixel geometries, experiment information. Secondly, it will start writing all dynamic measurement and event data based on provided flatbuffer schema, Kafka topics and sources.

The data writing will either stop after a preset amount of time or after the file writer receives an explicit stop job message. Once the file writer stops writing the dynamic data, it will proceed to generate some additional metadata, such as average, minimum, and maximum values of data sets written to the NeXus file. Once all this is done, the file writer will finally emit a writing done message to Kafka to notify that it is done writing a NeXus file and switch its state from busy to idle. An example of a file written in the detector tests is displayed in Fig. 8.

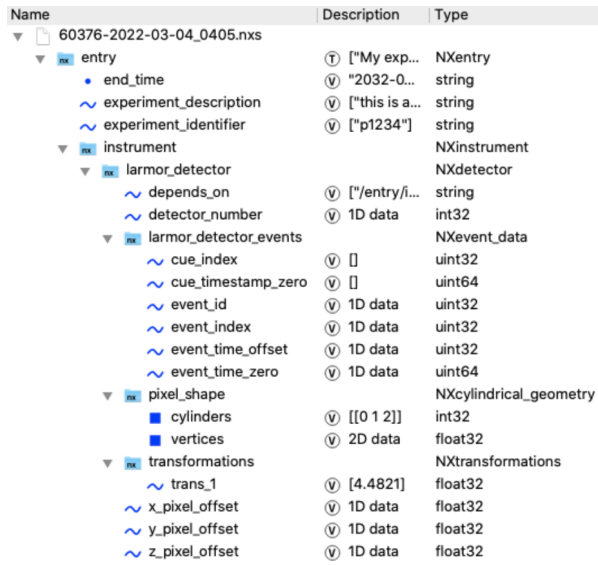


Figure 8: NeXus file with the structure written in the LoKI detector tests.

DATA REDUCTION WITH Scipp

Scipp is data reduction framework developed at ESS [8, 9]. It features handling of physical units, propagation of uncertainties, optimised data structure for multidimensional arrays, support for binned and event data, instrument visualisation, flexible masking and plotting. In order to provide flexibility, easy development and testing an overall scipp ecosystem consists of four main modules *scipp*, *scippnexus*, *scippneutron* and *ess* (Fig. 9). *scipp* is general package utilising multi-dimensional arrays of data with named dimensions and associated coordinates. *scippnexus* provides a link between the HDF5-based NeXus Data Format and *scipp*. *scippneutron* is specifically designed for handling neutron scattering data reduction by providing "unit conversions" and technique specific tools based on physics of (time-of-flight) neutron scattering. *ess* provides ESS facility and instrument bespoke tools for neutron data reduction and visualisation. All *scipp* modules are easily accessible from Jupyter-notebooks (*ess-notebooks*), which contain both code and documentation.

During LoKI detector test a few notebooks (at *ess-notebooks* level) were created primarily for debugging and

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR022

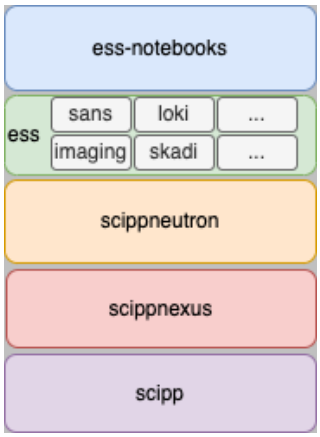


Figure 9: Scipp ecosystem architecture.

sanity checks. The main functionality of the notebooks involved: loading files, printing basic metrics (e.g number of recorded events), showing instrument view (Fig. 10) and plotting TOF spectra of detector and monitors. By performing such tests we were able to quickly identify issues with incorrect pixel positions and missing data.

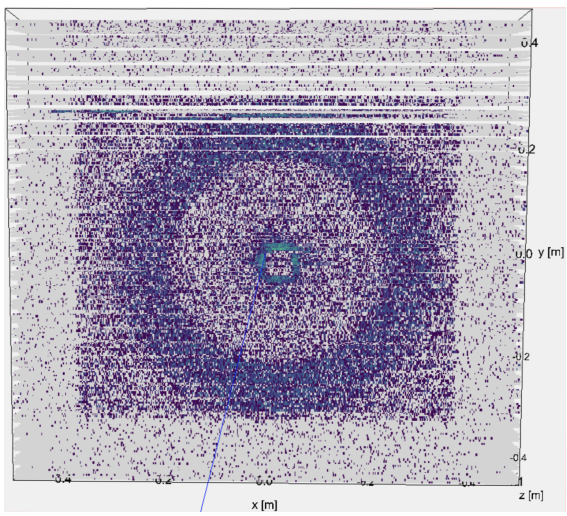


Figure 10: Silver behenate SANS data viewed with scipp's instrument view.

Once measurements were finished, the recorded data was further processed by performing series of calibrations and data reduction. The calibration data was collected by placing masked strips in front of the detectors with a uniformly scattering sample. The obtained peaks positions were used as a reference for further corrections and fitted using Mantid software [11] (it is anticipated that in future such corrections will be also performed in *scipp*). Corrected files were subsequently passed to *scipp* where data sets from different runs were merged. The merged files were subject of further data reduction protocol as demonstrated in (https://scipp.github.io/ess/techniques/sans/sans.html). In brief the protocol involves loading files, defining masking, applying coordinate corrections and then converting from TOF to lambda space and from lambda to Q space. Eventually background data is subtracted from sample and normalised (Fig. 11). The final outcome of data reduction protocol is intensity curve in the function of scattering vector, q.

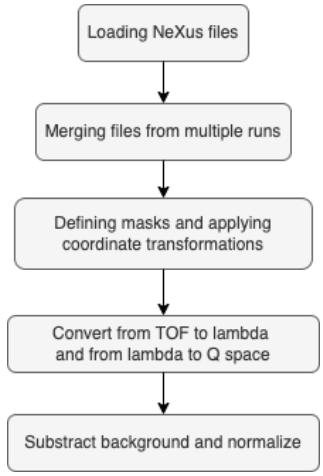


Figure 11: Data reduction workflow.

CONCLUSION

The LoKI test at ISIS in March 2022 demonstrated the functionality of the ESS data acquisition pipeline from the detector electronics, through FPGAs, into software, and into a state that is easily analysed and used for science. Furthermore, it demonstrated the ability to support the experiment remotely through remote diagnostics and debugging. The Readout Master aggregates detector readouts into packets, which are sent to the EFU to be processed into event data. The event data is sent to Kafka, where it is then used by the NeXus Filewriter to write a NeXus file per experiment, which is then analysed using scipp. This entire pipeline was running and live experiment data was processed in much the same way that it will be in production at the ESS facility.

REFERENCES

[1] K. Andersen *et al.*, "The instrument suite of the European Spallation Source," *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 957, p. 163 402, 2020. doi:10.1016/j.nima.2020.163402

[2] G. Náfrádi *et al.*, "Shielding considerations for ESS LoKI," *J. Neutron Res.*, vol. 22, no. 2-3, pp. 119–129, 2020. doi:10.3233/JNR-200152

[3] A. Mukai *et al.*, "Architecture of the data aggregation and streaming system for the European Spallation Source neutron instrument suite," *J. Instrum.*, vol. 13, no. 10, T10001, 2018. doi:10.1088/1748-0221/13/10/t10001

[4] S. Kolya *et al.*, "Building a research infrastructure and synergies for highest scientific impact on ESS, Deliverable 4.1 – Integration Plan for Detector Readout," 2015. doi:10.17199/BRIGHTNESS.D4.1

[5] M. Christensen *et al.*, "Software-based data acquisition and processing for neutron detectors at European Spallation Source—early experience from four detector designs," *J. Instrum.*, vol. 13, no. 11, T11002, 2018. doi:10.1088/1748-0221/13/11/t11002

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR022

[6] J. Kreps, N. Narkhede, and J. Rao, "Kafka: a Distributed Messaging System for Log Processing," 2011.

[7] *Daqlite*, 2022. https://github.com/ess-dmsc/daqlite

[8] S. Heybrock, O. Arnold, I. Gudich, D. Nixon, and N. Vaytet, "Scipp: Scientific data handling with labeled multi-dimensional arrays for C++ and python," *J. Neutron Res.*, vol. 22, no. 2-3, pp. 169–181, 2020. doi:10.3233/jnr-190131

[9] *scipp - Multi-dimensional data arrays with labeled dimensions*, 2022. https://scipp.github.io/

[10] M. J. Christensen and T. Richter, "Achieveing reliable UDP transmission at 10 Gb/s using BSD socket for data acquisition systems," 2017. doi:10.48550/ARXIV.1706.00333

[11] O. Arnold *et al.*, "Mantid—Data analysis and visualization package for neutron scattering and μ SR experiments," *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 764, pp. 156–166, 2014. doi:10.1016/j.nima.2014.07.029

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR023

EXPERIMENTAL DATA COLLECTION STANDARDS AT SESAME SYNCHROTRON

M. Alzubi*, A. Abbadi, A. Al-Dalleh, A. Aljadaa, A. Lausi, A. Mohammad, B. Aljamal, G. Iori, G. Kamel, M. Abdellatief, M. Genisel, M. Harfouche, R. khrais, S. Matalgah, Y. Momani
SESAME Synchrotron, Allan, Jordan

Abstract

Experimental data collection is the essential process of acquiring experimental raw data along with its associated metadata from SESAME beamlines. For data collection and processing; scanning modes, data and metadata formats, and data visualisation are only a few aspects in which individual beamlines differ from each other. In addition, the volume of experimental datasets every experimental day might range from a few gigabytes to many terabytes. Herein, the effectiveness of the experiments being conducted at SESAME depends heavily on the efficiency and reliability with which experimental data are collected. Each beamline at SESAME has its own Data Acquisition (DAQ) system that is intended and being developed primarily to meet beamline users' and scientists' expectations. It also ensures that experimental raw data and metadata are not randomly generated and are stored together in a stander and well-defined file formats in compliance with SESAME Experimental Data Management Policy. In this paper, we present the standards and features employed in SESAME's DAQ systems, as well as the experimental data creation, curation, storage, and accessibility pipeline currently being built for SESAME beamlines.

INTRODUCTION

SESAME [1] is a third-generation synchrotron light that has currently three beamlines in operation and serves the SESAME users' community; the XAFS/XRF (X-ray Absorption Fine Structure/X-ray Fluorescence) spectroscopy [2], MS (Materials Science) and IR (Infrared Spectromicroscopy) beamlines [3]. Two more beamlines are being installed and will be commissioned soon, namely, HESEB (HElmholtz-SEsame Beamline) [4] and BEATS (BEAmline for Tomography at SESAME) [5, 6].

At SESAME, the first monochromatic beam was obtained in November of 2017 at the XAFS/XRF beamline. In August of 2018, the first user experiment was conducted on the same beamline. In June of 2020, the SESAME Council adopted its "Experimental Data Management Policy," which is a deliverable of the H2020 BEATS project that is a part of the EU research and innovation funding programme. This policy is harmonised based on the European Synchrotron Radiation Facility (ESRF) and PaNData data policy frameworks [7]. Late in the same year, the organisational structure of SESAME was reformulated and a new dedicated team was created, under the supervision of the scientific director, to obtain the experimental data from the beamlines in compliance with data policy. In January of 2021, the Data Collection and Analysis (DCA) team was founded with these objectives: i) enabling the beamlines' DAQ systems to generate experimental data aligned with SESAME's Data Management Policy, ii) increasing the productivity of experiments, iii) enhancing the scanning time and quality, iv) considering the maximum integration levels with control, motion, and scientific computing systems, v) minimising user experience gaps, vi) automating raw data and metadata collection, and vii) adhering to software engineering standards developing systems inasmuch as possible.

DATA POLICY AND DAQ STANDARDIZATION

The implementation of the data policy not only aids standardising DAQ systems and the data sets generated across beamlines, but also enhances the integration between DAQ and other systems in which they serve as a source of raw data and metadata, as well as the computing infrastructure used to store data and provide access and analysis services on it.

In nutshell, the data policy constitutes and describes the ownership, storage, access and management of the experimental data generated from SESAME beamlines [8]. It is applied on the experimental data for both users awarded beamtime proposals and in-house experiments. There is a three year embargo period after an experiment is completed, during which only the principal investigator of the beamtime proposal and the experimental team have access to the data. Passing the three years, the experimental data becomes openly accessible to SESAME registered users. There is a possibility to exceptionally extend this period by submitting a request to SESAME officials. The experimental data is kept at SESAME archiving for at most ten years on best efforts basis; the exact number of years will be defined for each beamline in reference to the data type and volume considering the financial and technical limitations. The policy implies sorting and archiving the experimental data in well-defined format where such data should be acquired from SESAME software and associated with unique key identifier. In contrast, this data policy does not apply to industrial research and collaborative research with SESAME scientists where other policies will be specifically developed.

EXPERIMENTAL DATA COLLECTION

The majority of SESAME's beamlines use locally developed and maintained DAQ systems. They are the main

* mostafa.zoubi@sesame.org.jo

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR023

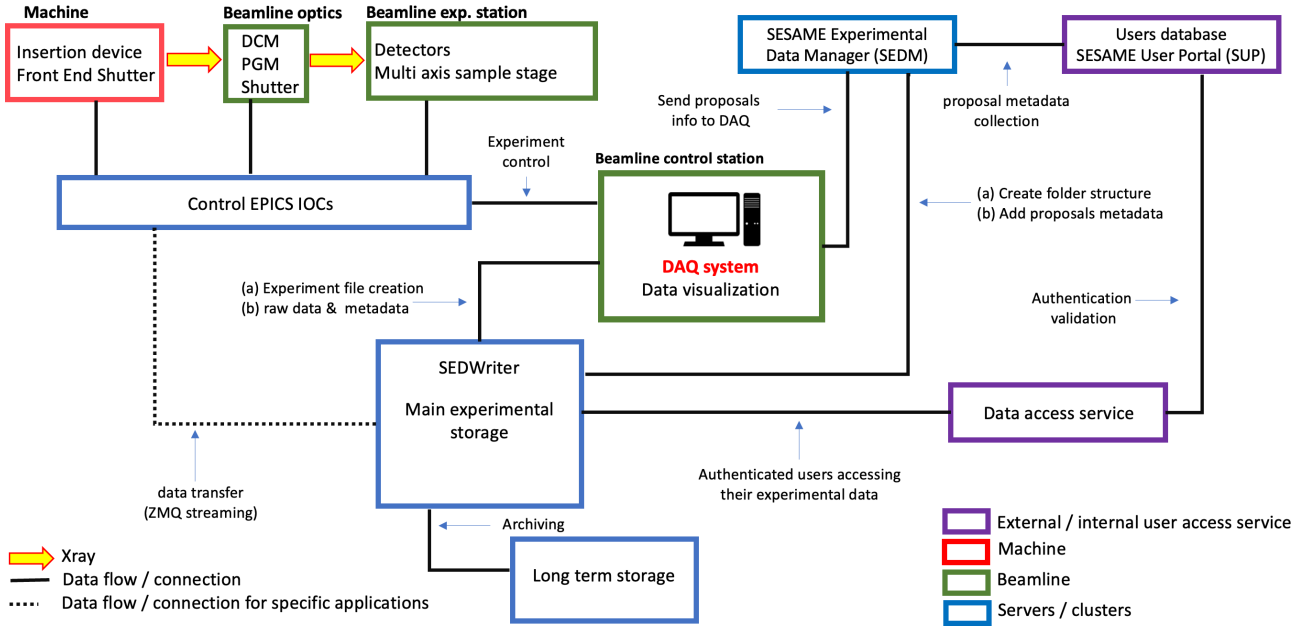


Figure 1: A General block diagram describes beamlines DAQ system at SESAME and its interaction with other systems. Normally, Double Crystal Monochromator (DCM) and Plane Grating Monochromator (PGM) are not combined together in the same beamline also not all beamlines have Insertion Device (ID) as a source, however, this indicative block diagram shows different beamline components that are controlled or provide data to DAQ through the Experimental Physics and Industrial Control System (EPICS) Input Output Controllers (IOCs) layer. On the other hand, it is very common to have more than one detector in the beamline where DAQ is supposed to drive them either individually or simultaneously depending on the beamline and applications.

beamline users' and scientists' interface to set up and control scans for acquiring and collecting experimental data. Each beamline at SESAME has its own and specific DAQ system, which is pure object-oriented and follows the modern software engineering development standards in terms of modularity, testing and validation, user experience, documentation, and code version control. Figure 1 shows a general block diagram that describes the DAQ pipeline. The DAQ system interacts with many other systems to create experimental data and makes it available for access, processing, and archiving services.

Beamline Scanning Techniques

All of the beamlines that use our DAQ system now make use of step scanning technique, which is easy to use, reliable, and operable even with basic hardware and control support. The main drawback with this technique is the long time it takes to complete a scan, as most of the time is spent waiting for mechanical parts to move and, once the movement is done, for mechanical vibration to settle [9, 10]. Moreover, detector readout time and the DAQ system's connection time are additional elements that impose delays. In order to get around this issue, several synchrotron facilities have introduced continuous scanning mode namely the on-the-fly mode. This technique has been successfully demonstrated and put into operation with a large number of beamlines across the facilities.

At SESAME, recent work carried out on the testing bench of the BEATS beamline, resulted in the development of a software-based continuous scan (no external triggering system). This was done in an attempt to synchronise the detector readout frames with the rotating stage speed. Visible light tomography [11] has been used to evaluate the effectiveness of the technique as the beamline is being constructed, and the results have been very encouraging.

DAQ Standers and Features for Beamlines

SESAME's DAQ system for any beamline is constructed on top of the EPICS control system. This necessitates the presence of EPICS device support for the various hardware components that are placed at the beamlines, the control team also employs this as their standard when installing new devices and components at SESAME. The DAQ system also employs standard EPICS records and drivers for certain and often used subsystems. More specifically, EPICS motor records are used for motion system, while areaDetector driver is used for any 2D imaging detector [12].

Graphical User Interface (GUI) is used for the DAQ system which helps the users to easily and quickly understand the experimental settings and the scanning parameters. Motif Editor and Display Manager (MEDM), EPICS Qt, and PyQt are now being used as GUI clients throughout the beamlines.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR023

Automated energy calibration module has been recently added to DAQ of the XAFS/XRF beamline. The module adjusts the monochromator's Bragg angle (theta) based on selected and well known metallic references of some elements . The module enables users to perform a calibration scan, from which necessary data is automatically extracted and made available to different processing techniques via a GUI, before proceeding to the final step, which calculates the angular offset of and set it tothe theta motor as a correction between Bragg angle and the energy.

In order to ensure that all DAQ systems are in line with the data policy, the SESAME users' database should be integrated with all of them. The database contains metadata about users and their submitted beamtime proposals. In this context, recently, we were able to effectively link the XAFS/XRF beamline DAQ system with the database by means of a module that is developed and named SESAME Experimental Data Manager (SEDM). On each experimental day, this module pulls out all metadata pertaining to accepted and scheduled proposals, creates folder structures includes the experimental data path for each proposal in the main storage using a predefined naming convention scheme, and then sets the appropriate access and read/write permissions for those folder structures. SEDM can intelligently identify and communicate proposals' metadata to their assigned beamlines, allowing DAQ to inject this data into the experimental files. Before each scan, the DAQ system verifies all scan parameters to prevent human mistakes, with proposal number validation being the most critical, as it is used to uniquely identify the generated data sets. For beamtime allocated to accepted proposals, users are required to enter their proposal number to be validated with the SEDM metadata.

Figure 2 shows the validation process in an activity diagram. The system only accepts scheduled proposals for current experimental day, otherwise users must contact the beamline scientist, who has full access to the beamline calendar, in order to reschedule an accepted proposal on this particular experimental day.

Since unanticipated issues can crop up during scans, our DAQ system supports for what we call unattended scan, which implies that a scan may run, detect issues, and take corrective action without human intervention. Currently, one action has been implemented that the system does not acquire noise data when the readouts of beamline-specific specified parameters exceed predefined limits (i.e. beam availability and energy, shutters, detector readout, .. etc). In such scenario, the scan will be temporarily paused and resumes automatically when favourable conditions are satisfied.

The system is able to automatically switch between samples, each of which can have different scan parameters applied. The number of samples is dependent only on the capability of the sample holder used in the beamline and has no software limits. The user is responsible for assigning the name and position to identify each sample. The naming convention that DAQ adopts for generating experimental files necessitates that the name shall be descriptive of the

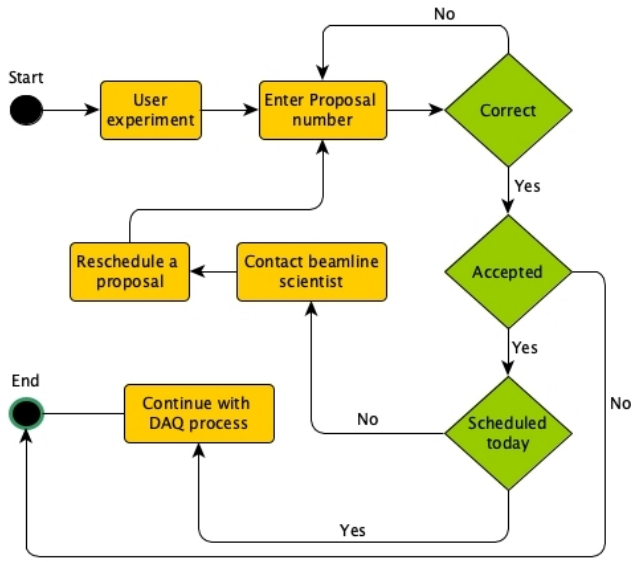


Figure 2: Activity diagram illustrates the proposal validation process on the experimental day before start collecting data sets.

sample. In most cases, the number of experimental files is typically proportional to the number of samples.

A live data visualisation module has been added to the DAQ system, allowing users to see live raw data or live data representations in various graphs. The module also shows scan progress statistics, and includes basic analysis functions.

The DAQ system coordinates and tracks experimental data flow from its origins (IOCs) to its destination in the main storage. The dataflow route can either go through the DAQ itself or direct streaming from the IOCs to the storage, with the choice depending on the dataset volume size being sent, the Operating System (OS) hosting the IOC driver, and the application needs for a particular beamline. Experimental data goes through DAQ at most of the beamlines. Recently, we implemented a direct streaming solution on BEATS testing bench, which streams the data from the detector driver to the main storage at 1.1 Gigabyte per second, avoiding the limitations imposed by protocols transferring bigdata between different files systems. The publisher-subscriber sockets connection type of the asynchronous messaging library ZeroMQ (ZMQ) was used achieving this solution [5].

On the main storage, the customized module we built, SESAME Experimental Data Writer (SEDWriter), is responsible for writing the experimental data set into experimental files including raw data and metadata. In order for SEDWriter to function, it needs to know the origin of this data, type of experiment whether in-house or user experiment, the file format it has to create and some other beamline-specific settings. The needed information to the writer is provided by DAQ system of each beamline. At the time being, our writer generates experimental data files in three formats, namely X-ray absorption spectroscopy Data Interchange (XDI), Hierarchical Data Format version 5 (HDF5) - Scientific Data Exchange (DXfile) and data file (DAT). XDI is the stander

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR023

experimental file for XAFS/XRF and HESEB beamlines. It is an open-source ASCII-based file that can be opened in any text editor, self-documented, flexible with adding metadata, and is being developed and used in similar laboratories [13]. The DXfile format will be utilised to record experimental data for BEATS beamline and it has been successfully tested on its testing bench. The DXfile file is open-source HDF5 based, self-describing with a tree structure, and supported in the file plugin of areaDetector driver. The file includes a full description about the proposal, experiment, beamline, sample, sample stage and detector [14].

On the other hand, to be in compliance with our data policy standardising SESAME experimental data files, we are evaluating DXfile format to be adopted as a universal experimental file format for all beamlines at SESAME. Diffraction data and other metadata associated with the MS beamline are recorded in DAT files.

Every time an experiment is run using our DAQ system, all the scan parameters, choices, and settings are recorded in plain-text configuration file. This configuration file is sent to SEDWriter so that any available metadata may be extracted, and the file itself can be used again to import its contents into DAQ GUI for editing to run another scan.

Within our system, we have implemented a simple online logging module that provides both a timestamp and colour labelling for each entry. This functionality assists us in error troubleshooting, makes it simpler to debug, and allows us to track which part of DAQ is now being executed. The logs are also captured in a log file for each experiment, and this log file is stored next to the experimental data files so that users and scientists have access to them in the event that they are required.

With each DAQ system that we provide for a beamline, we used to write a straightforward documentation that is aimed at the scientist who would be operating the system. We recently made the switch to a professional open-source documentation service, namely "Read the Docs" that caters to beamline scientists, users, and developers. This was necessitated by the fact that we are a users' facility and also utilise GitHub for version control, which is already integrated with "Read the Docs".

DATA ACCESS AND ARCHIVING

Each beamline's DAQ system has distinct needs for storing experimental data. From storage administrator standpoint, the dataset volume, the number of files, and the desired writing speed performance define out our storage solutions. Currently, the primary experimental storage system is comprised of two high-end disc arrays: i) General Parallel File System (GPFS) and ii) Storage Area Network (SAN) storages. The Server Message Block (SMB)/Common Internet File System (CIFS) and Network File System (NFS) protocols make both arrays accessible for services, users and beamline scientists. The access and permissions are managed by SESAME's active directory server. On the other hand, for cluster-to-cluster access or clients with high throughput demands, we use the native GPFS client, which allows transferring data at the maximum performance of the storage.

Access to experimental data is a vital aspect of completing the experiment life cycle, where dataset exporting activities are performed at two levels to give all possible data manipulation options for beamline scientists and users. On the basic level, the generated experimental data is downloadable by an authorized access service using SMB and NFS clients. This level is designed for on-premises use and the transmission of relatively modest datasets. For the advanced level, we are currently implementing ICAT [15] service, which is an open-source large-scale data management system, developed and maintained by scientific communities, and it has all necessary features needed to securely enable the web access to our experimental data. ICAT implementation will be focused on adapting SESAME's data policy, it is already a data catalogue service and the core elements of FAIR data (findability, accessibility, interoperability, and re-usability) are already there. ICAT service will be integrated with SESAME storage systems and the universal experimental file format that will be used. Through ICAT, the users will also be able to post analysed results from their experiments and link those results to their publications using the service.

Undoubtedly, the fast growth of experimental data as well as the adoption of the SESAME data policy will make it necessary to establish a long-term data life cycle service. To achieve this, a data archiving solution should be serving in place. In accordance with our data policy, this will be the primary location where experimental data will be stored for a period of 10 years. At the moment, we are evaluating different technologies to implement this service. Cloud, disc, and tape based storages are compared from a variety of perspectives, with emphasis on reliability, data availability, and initial and running costs. Early findings suggest that tape-based storage might be the optimal solution for us.

CONCLUSION

This paper describes our DAQ systems, standards, and procedures for the majority of SESAME beamlines. Not one beamline's DAQ in particular is addressed in any great detail. Table 1 provides a summary of available and planned DAQ system features. All systems are operational and fulfil the needs for the time being. However, for the upcoming period, it will be essential to implement continuous on-the-fly scanning mode in order to significantly reduce the scan times. Given that BEATS will be the first massive data-generating beamline at SESAME, it is advantageous to have a data policy in place prior to the expected increase in data volumes. As a result, the DAQ systems on all beamlines, including BEATS, are currently confirmed to a set of standards to assure policy compliance.

ACKNOWLEDGEMENTS

We thank the EU-H2020 BEATS project and European Synchrotron Radiation Facility (ESRF) for sharing assisting and helping us developing our data policy.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022-FR023

Table 1: DAQ Features Implementation Summery/Plan: Completed (C) – Under Development (D) – Planned/Under evaluation (P) – To be improved (T) – Not Applicable (NA)

Feature	XAFS/XRF	MS	HESEB	BEATS
EPICS motor records	C	C	C	P
areaDetector	NA	D	NA	C
GUI	C	P	C	C
Auto energy calibration	C	T	T	NA
Users' DB integration	C	D	C	P
Step scan	C	C	C	C
Cont. scan	P	P	P	C
Unattended scan mode	C	C	C	P
Auto sample changing	T	P	T	P
Data visualisation	T	T	C	C
ZMQ streaming	NA	D	NA	C
Common file format	C	P	C	C
Exp. config file	C	C	C	C
Logging	C	C	C	C
Code version control	C	C	C	C
Documentation	D	P	C	P
Feature	**Institution level, for all beamlines**			
Universal file format	P (dxFile as initial finding)			
Long-term archiving	P (tape-based storage as initial finding)			

REFERENCES

[1] SESAME Synchrotron-light, https://www.sesame.org.jo

[2] M. Harfouche, M. Abdellatief, Y. Momani, A. Abbadi, M. Al Najdawi, M. Alzubi, B. Aljamal, S. Matalgah, Lu. Khan, A. Lausi, and G. Paolucci, "Emergence of the first XAFS/XRF beamline in the Middle East: providing studies of elements and their atomic/electronic structure in pluridisciplinary research fields", *J. Synchrotron Radiat.*, vol. 29, pp. 1107-1113, 2022. doi:10.1107/S1600577522005215

[3] G. Kamel, S. Lefrancois, T. Moreno, M. Al-Najdawi, Y. Momani, A. Abbadi, G. Paolucci, and P. Dumas, "The first infrared beamline at the Middle East SESAME synchrotron facility", *J. Synchrotron Radiat.*, vol. 28, pp. 1927-1934, 2021. doi:10.1107/S1600577521008778

[4] W. Drube, MF. Genisel, and A. Lausi, "SESAME Gets Soft X-Ray Beamline HESEB", *Synchrotron Radiat. News*, vol. 35, iss. 1, pp. 22-22, 2022. doi:10.1080/08940886.2022.2043710

[5] G. Iori, S. Matalgah, C. Chrysostomou, A. Al-Dalleh, and M. Alzu'bi, "Data Acquisition and Analysis at the X-ray Computed Tomography Beamline of SESAME", *IEEE Jordan Inter. Joint Conf. Electr. Eng. Inf. Technol. (JEEIT'21)*, pp. 134-139, 2021. doi:10.1109/JEEIT53412.2021.9634151

[6] M. Abdellatief, M. A. Najdawi, Y. Momani, B. Aljamal, A. Abbadi, M. Harfouche, and G. Paolucci, "Operational status of the X-ray powder diffraction beamline at the SESAME synchrotron", *J. Synchrotron Radiat.*, vol. 29, pp. 532-539, 2022. doi:10.1107/S1600577521012820

[7] R. Dimper, A. Götz, A. de Maria, V.A. Solé, M. Chaillet, and B. Lebayle, "ESRF Data Policy, Storage, and Services", *Synchrotron Radiat. News*, vol. 32, iss. 3, pp. 7-12, 2019. doi:10.1080/08940886.2019.1608119

[8] SESAME Experimental Data Management Policy, https://www.sesame.org.jo/for-users/user-guide/sesame-experimental-data-management-policy

[9] Shang-Wei Lin, Chia-Feng Chang, Robert Lee, Chi-Yi Huang, Chien-I Ma, Liang-Jen Fan and Hok-Sum Fung, "On-the-fly Scan: Improving the Performance of Absorption Spectrum Measurement", in *Proc. 11th Int.l Conf. Synchrotron Radiat. Instrum., J. Phys.: Conf. Ser.*, vol. 425, p. 122002, 2013. doi:10.1088/1742-6596/425/12/122002

[10] S. Zhang, Y.M. Abiven, J. Bisou, G. Renaud, G. Thibaux, F. Ta, S. Minolli, F. Langlois, M. Abbott, T. Cobb, C.J. Turner "Pandabox: A Multipurpose Platform for Multi-Technique Scanning and Feedback Applications", in *Proc. 16th Int. Conf. on Accelerator and Large Exp. Control Sys. (ICALEPCS'17)*, Barcelona, Spain, 2017. doi:10.18429/JACoW-ICALEPCS2017-TUAPL05

[11] LINXS, https://www.linxs.se/news/2021/10/11/from-kitchen-tomography-to-a-cutting-edge-neutron-imaging-beamlinenbsp

[12] EPICS, https://epics.anl.gov

[13] XAS Data Interchange, https://github.com/XraySpectroscopy/XAS-Data-Interchange

[14] DXfile, https://dxfile.readthedocs.io/en/latest/

[15] D. Flannery, B. Matthews, T. Griffin, J. Bicarregui, M. Gleaves, L. Lerusse, R. Downing, A. Ashton, S. Sufi, G. Drinkwater, K. Kleese, and D. Limited, "ICAT: Integrating data infrastructure for facilities based science", 2009 Fifth IEEE International Conference on e-Science, Oxford, UK. doi:10.1109/e-Science.2009.36

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022

List of Authors

Bold papercodes indicate primary authors; ~~strike through/black~~ papercodes indicate **no submission received**

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022

Institutes List

ALBA-CELLS
Cerdanyola del Vallès, Spain
- Agostini, G.
- Avila-Abellan, J.A.
- Baucells, A.
- Escudero, C.
- Matilla, O.
- Rubio-Manrique, S.
- Serra-Gallifa, X.
- Vallcorba, O.

ANL
Lemont, Illinois, USA
- Bunnell, K.J.
- Dickerson, C.
- Nardi, B.G.
- Novak, D.J.
- Stanton, D.

Aquenos GmbH
Baden-Baden, Germany
- Marsching, S.

CERN
Meyrin, Switzerland
- Abdulhalim, A.
- Gkotse, B.
- Mølholm, A.S.
- Ravotti, F.

Cosylab
Ljubljana, Slovenia
- Jesenko, A.

COSYLAB, Control System Laboratory
Ljubljana, Slovenia
- Debenjak, A.
- Marn, M.
- Oven, Ž.
- Pajor, G.
- Varlec, J.

DESY
Hamburg, Germany
- Bacher, R.
- Branlard, J.
- Delfs, T.
- Duval, P.
- Eichler, A.
- Fröhlich, L.
- Hensler, O.
- Jastrow, U.
- Meykopff, S.
- Szczesny, J.
- Tempel, T.
- Timm, J.H.K.
- Walla, M.
- Wilgen, J.
- Wilksen, T.

DESY Zeuthen
Zeuthen, Germany
- Melkumyan, D.
- Weisse, S.

Dimtel
Redwood City, California, USA
- Teytelman, D.

E9
London, United Kingdom
- Klys, K.
- Lukaszewski, M.

Elettra-Sincrotrone Trieste S.C.p.A.
Basovizza, Italy
- Bogani, A.I.
- Pivetta, L.

ELI-BEAMS
Prague, Czech Republic
- Capuano, F.
- Mazanec, T.
- Peceli, D.
- Plötzeneder, B.
- Rus, B.
- Špaček, A.

EMBL
Hamburg, Germany
- Blanchet, C.
- Fiedler, S.
- Galikeev, E.G.
- Jahn, D.
- Meyer, J.
- Palnati, V.
- Ristau, U.
- von Stetten, D.

ESS
Lund, Sweden
- Alcock, S.
- Christensen, M.J.
- Houston, J.E.
- Muric, K.
- Potrzebowski, W.
- Richter, T.S.
- Walker, J.L.

EuXFEL
Schenefeld, Germany
- Baranasic, B.
- Bueno, M.
- Coppola, N.
- Feltrin Zanellatto, L.
- Freyermuth, T.
- Gessler, P.
- Giovanetti, G.
- Hauf, S.
- Huynh, S.T.

ISBN: 978-3-95450-237-0 ISSN: 2673-5512 doi:10.18429/JACoW-PCaPAC2022

- Jardón Bueno, N.
- Mashayekh, N.
- Silenzi, A.
- Stupar, M.
- Teichmann, M.
- Tolkiehn, J.

Fermilab
Batavia, Illinois, USA
- Harrison, B.F.

GANIL
Caen, France
- Delahaye, O.
- Haquin, C.H.
- Patard, C.H.
- Pillon, F.
- Pivard, J.
- Sénécal, G.

GSI
Darmstadt, Germany
- Dziewiecki, M.
- Fitzek, J.
- Hüther, H.C.
- Kaiser, K.
- Lüghausen, K.
- May, G.M.
- Mueller, R.
- Peter, B.
- Rauch, S.
- Schaller, A.
- Thieme, M.
- Walter, A.

HZDR
Dresden, Germany
- Justus, M.
- Steinbrück, R.
- Zenker, K.

J-PARC, KEK & JAEA
Ibaraki-ken, Japan
- Kamikubota, N.
- Sato, K.C.
- Yang, M.

JAEA/J-PARC
Tokai-mura, Japan
- Kikuzawa, N.

KIS
Ibaraki, Japan
- Tajima, Y.

KIT
Karlsruhe, Germany
- Blomley, E.
- Gethmann, J.
- Mexner, W.
- Mochihashi, A.
- Müller, A.-S.
- Schreiber, P.
- Schuh, M.

MINES ParisTech, PSL Research University
Paris, France
- Jouvelot, P.

Politecnico di Torino
Torino, Italy
- Tiboni, G.

S2Innovation
Kraków, Poland
- Gandor, M.
- Goryl, P.P.
- Kowalczyk, J.T.
- Madej, T.M.
- Nabywaniec, M.
- Żytniak, Ł.

SESAME
Allan, Jordan
- Abbadi, A.
- Abdellatief, M.
- Abugharbiyeh, M.M.
- Al-Dalleh, A.
- Aljadaa, A.
- Aljamal, B.R.
- Alzubi, M.A.
- Genisel, M.F.
- Harfouche, M.
- Iori, G.
- Kamel, G.S.
- Khrais, R.
- Lausi, A.
- Matalgah, S.A.
- Mohammad, A.S.
- Momani, Y.R.

SSRF
Shanghai, People's Republic of China
- Chen, G.H.
- Lv, H.H.
- Mi, Q.R.
- Yan, Y.B.

STFC/RAL/ISIS
Chilton, Didcot, Oxon, United Kingdom
- Raspino, D.

VECC
Kolkata, India
- Chaddha, N.
- Pal, S.
- Roy, A.
- Sahoo, S.