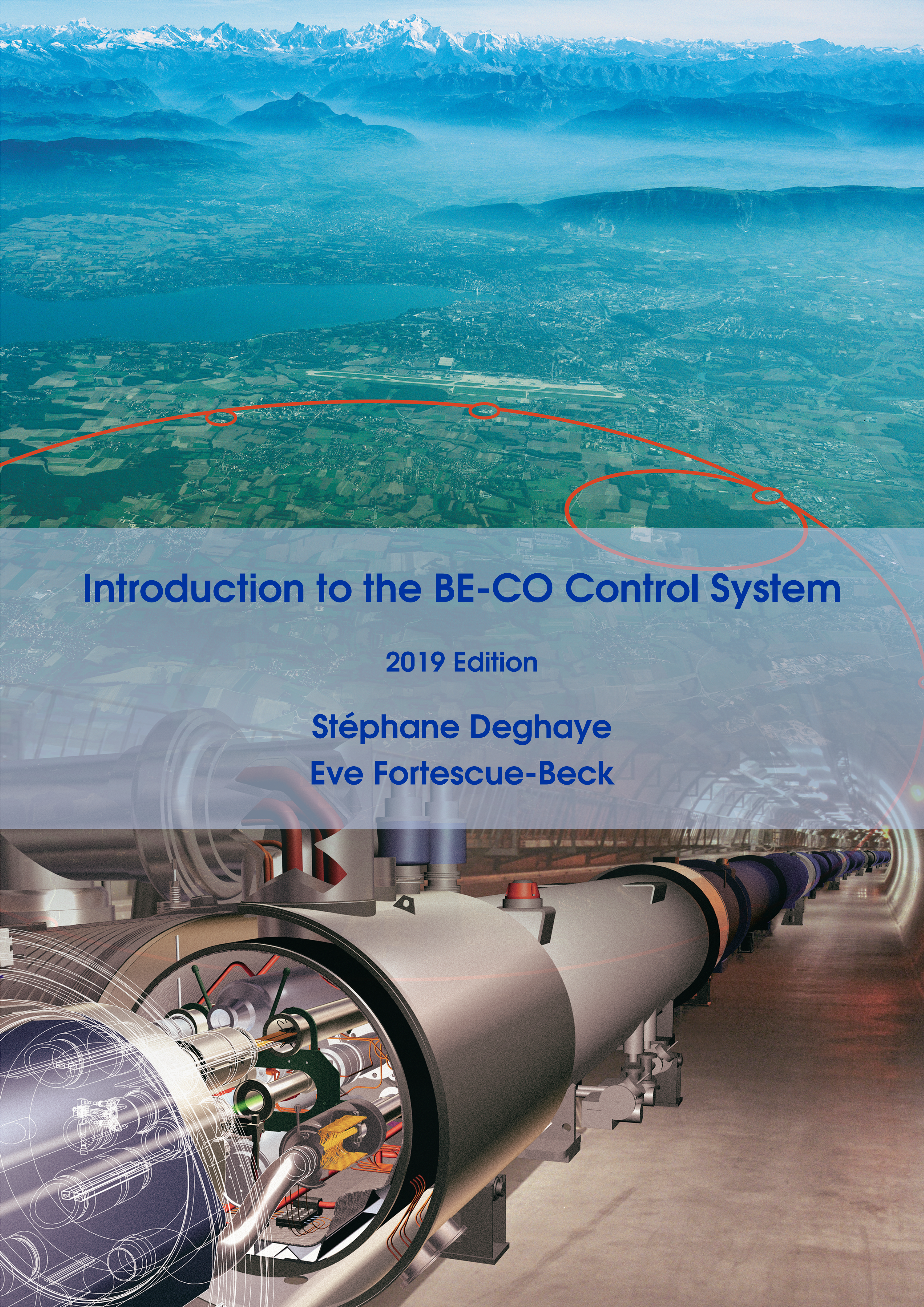


Introduction to the BE-CO Control System

2019 Edition

Stéphane Deghaye
Eve Fortescue-Beck

First edition, July 2017
This edition, January 2020

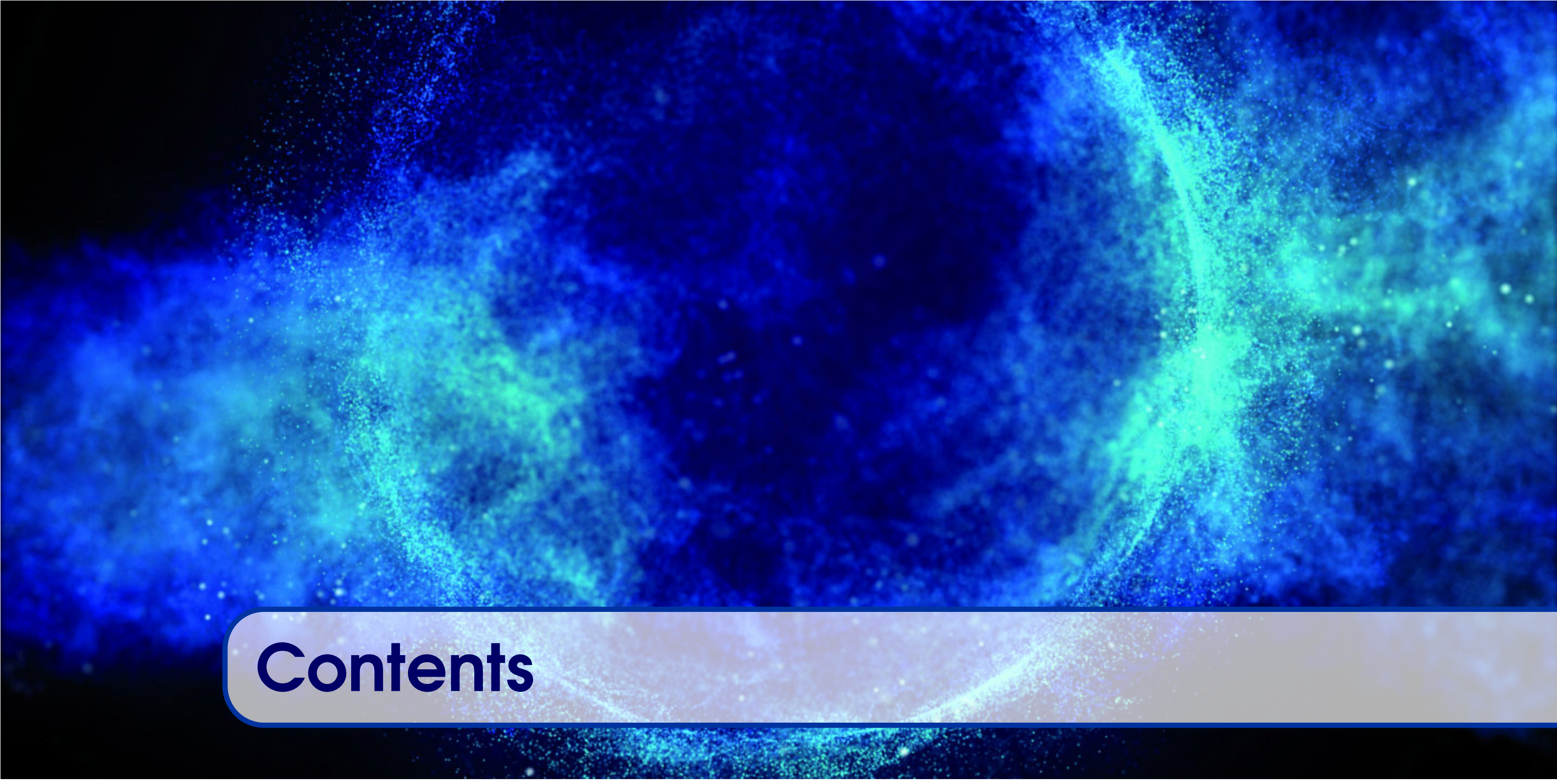
ABSTRACT

This document is an attempt to fill the gap between the Control System, as provided by the Controls (CO) group, and anybody who needs to work with the Control System. Providing the glue between the particle accelerator physicists and the accelerators' equipment, the Control System is built on a 3-tier architecture made of hundreds of physical nodes and tens of services. After a short introduction of the different physical tiers and of two basic use cases (control of setting values and monitoring of accelerator variables), we cover two core topics, the device-property model and timing. Next, the reader takes a complete bottom-up tour through the controls infrastructure, discovering the technologies used and the architectures in place. In order to give concrete examples, the main applications of this infrastructure are also covered at the end of the document.

Acknowledgements

The authors would like to thank the following persons for their valuable input and support during the writing of this document:

Michel Arruat, Vito Baggiolini, Jean-Claude Bau, Alastair Bland, Lukasz Burdzanowski, Mark Buttner, David Cobas, Claude Dehavay, Felix Ehm, Philip Elson, Mathieu Gabriel, Luigi Gallerani, Jean-Christophe Garnier, Enzo Genuardi, Tristan Gingold, Roman Gorbonosov, Eva Gousiou, Eugenia Hatziangeli, Frederic Hoguin, Greg Kruk, Dimitris Lampridis, Frank Locci, Stephen Page, Maciej Peryt, Anastasiya Radeva, Chris Roderick, Eric Roux, Pascal Le Roux, Javier Serrano, Ivan Sinkarenko, Wojtek Sliwinski, Marcin Sobieszek, Tom Włostowski, Adam Wujek



Contents

Introduction

This document is an attempt to fill the gap between the Control System as provided by the BE-CO group within the Beams (BE) department and anybody who needs to work with the Control System. You might have to refresh your understanding, or keep your knowledge up to date in order to take some expert decisions. You might be a newcomer joining the Controls group or part of the management. The idea behind the document is to allow the reader to create a map of the different sub-systems and components which form the BE-CO Control System and how they interact to fulfil the system's role: the control and monitoring of the accelerators. Last but not least, the document should also allow the authors to keep up to date with the huge number of components, products and services provided by BE-CO.

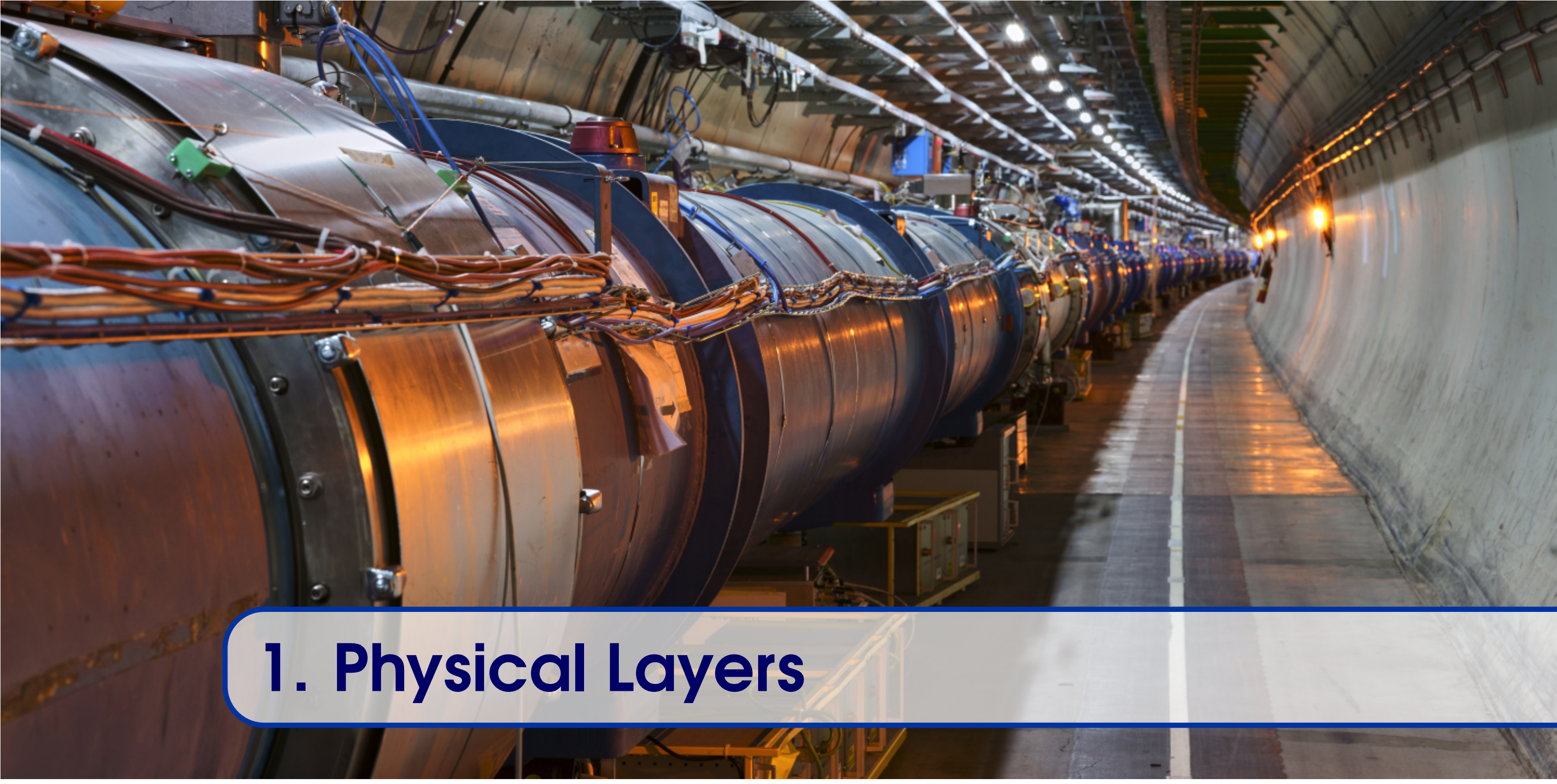
We aim to cover all of the components, products and services provided by the Controls group. But the effort made to control our accelerators could not be done in isolation and there is a partnership between the Controls group and other partner groups such as the Operations (OP), the Beam Instrumentation (BI) in the BE department, the Survey Mechatronics Measurements (SMM) and the Electrical Power Converters (EPC) groups from the Engineering (EN) and the Technology (TE) departments respectively. As much as possible, the components, products and services provided by our partner groups are cited for completeness but not described. This can be seen as an incomplete description as the Controls group is mainly an infrastructure provider but the endeavour would be too big for a team of two writers. This document is our view of the Control System as we understand it from our discussions with the people responsible for each component. It might not be the way the project leaders would have explained it but we hope it is factually correct and understandable for the target audience.

We start our adventure with a high-level overview of the core concepts, a description of the physical layers and typical data flows and operations. The reader who is only interested by an executive summary of the Control System could stop at this point. Then, in parts II to VII, we take the reader on a bottom-up trip and look at all the infrastructure components we provide. Readers willing to dig deeper may selectively read the chapters on specific components, and the bravest could read them all. In many cases, the Controls group acts as a user of the controls infrastructure, therefore in part VIII we study two examples of full-stack applications of the Control System. The document is heavily cross-referenced so that information can be accessed from many different entry points. In addition to the table of contents at the start of the document, in part IX, there is a table of figures, a list of acronyms, a glossary, and an index. All have page numbers referencing where the entries can be found in the text. The glossary focuses on non-CERN specific computer science terms, and CERN-specific terms not related to controls. There is also a bibliography for further reading.

Why a Control System?

Before even looking at how the CERN accelerators' Control System is designed, we should ask ourselves: Why do we need a Control System at all? Here are some explanations in order to put the remainder of the document into perspective. Particle accelerators are made of many components to control and monitor the beams produced. Those elements can be grouped into sub-systems. To cite a few, there are power converters to feed the magnets, which bend and focus the beam, RF amplifiers and cavities to accelerate and bunch together the particles, instruments to measure the beam's characteristics such as its position, its profile and many others. In turn, the physicists and operators need to be able to remotely control and monitor these elements; this is the role of the Control System. Placed between the operators and the accelerator hardware, the Control System's job is to set reference values (aka settings) and states in active elements (e.g. power converters), to read instruments, to monitor the health of sub-systems, and to diagnose faults, etc.

High-Level Concepts



1. Physical Layers

When looking at the Control System as a whole, as shown in figure 1.1, one can identify three physical layers and, therefore, we can describe the Control System as having a 3-tier architecture. The top (or client) tier is a set of computers used by the operations teams and equipment experts to run high-level graphical applications. The middle (or business) tier is made of powerful servers running the server side of the high-level applications. The lower (or front-end) tier is composed of embedded computers running real-time applications, interacting with electronic boards to control and monitor the accelerator components.

1.1 Control-Room Computers

The control rooms' computers are often referenced to as "consoles". The consoles run high-level graphical applications (aka Graphical User Interfaces (GUIs)) that interact with the other parts of Control System. The control-room consoles are Linux-based desktop PCs and the vast majority of the GUIs are written in Java using the Swing graphical toolkit. The users interact with the consoles via traditional means i.e. keyboard, mouse and one or several screens as shown in figure 1.2. More information on the consoles can be found in section 6.2.

In addition, for the operators working in the CERN Control Centre (CCC), which is the main control room, wall screens displaying live status information about the accelerator complex are installed (see section 17.1.3).

1.2 Back-End Servers

The back-end servers are rack-mountable, multi-Central Processing Unit (CPU) PCs tailored for 24/7 operation. Typically installed in the Controls Computer Room (CCR) next to the CCC, these Linux servers are file servers or run the server-side or business logic of the

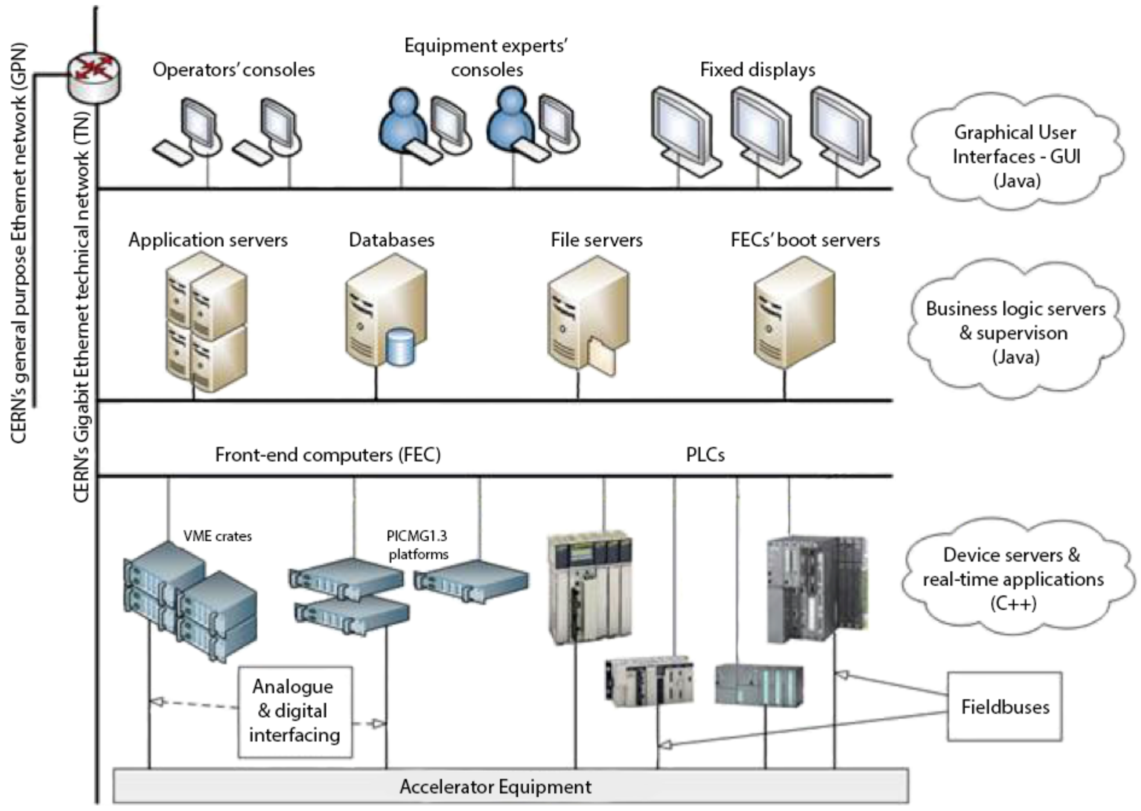

CERN's general purpose Ethernet network (GPN)
CERN's Gigabit Ethernet technical network (TN)
Operators' consoles
Equipment experts' consoles
Fixed displays
Graphical User Interfaces - GUI (Java)
Application servers
Databases
File servers
FECs' boot servers
Business logic servers & supervison (Java)
Front-end computers (FEC)
PLCs
VME crates
PICMG1.3 platforms
Device servers & real-time applications (C++)
Analogue & digital interfacing
Fieldbuses
Accelerator Equipment

Figure 1.1: The Control System's three-tier architecture

Figure 1.2: CERN Control Centre's consoles and wall screens

high-level applications. Most of the server processes are written in Java and communicate with the GUIs using Java-specific protocols such as Remote Method Invocation (RMI) and Java Message Service (JMS). Figure 1.3 shows an enclosure (standard 19-inch rack-mountable) with 14 servers. More details on the servers and the operating system they run can be found in section 6.1 and chapter 8 respectively. Furthermore, the control applications deployed on the application servers are discussed in part VIII.

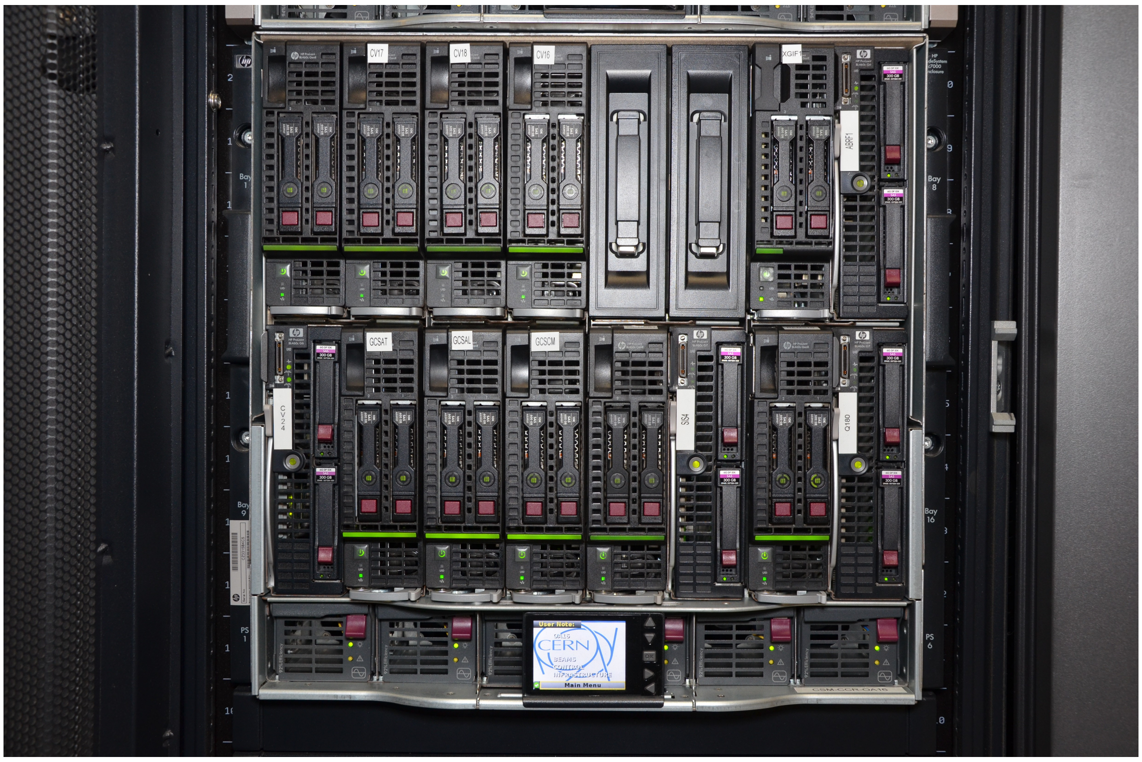


Figure 1.3: 19-inch enclosure with 14 servers running the controls business logic

1.3 Front-End Computers

The Front-End Computers (FECs) are also rack-mountable chassis based on different industrial standards such as Versa Module Europa (VME), PCI Industrial Computer Manufacturers Group (PICMG) 1.3, Compact Peripheral Component Interconnect (PCI), etc. For reliability reasons, they are single-board systems without screen, keyboard or hard drive; they only contain a CPU, memory and interfaces (network and bus bridge). In 2008, we decided to shift to the INTEL architecture and Linux and renovate all of the installations based on PowerPC CPUs running LynxOS. In 2019, we eventually finished upgrading of all of the FECs. The main purpose of the FECs is to perform the low-level real-time control and acquisition of the accelerator hardware. Figure 1.4 shows a VME crate containing a CPU board (far left), two timing receivers and 4 analogue function generators (2 fast and 2 slow). Readers interested by the hardware aspects of the front-end computers can find out more in chapter 5. The software side, operating system, kernel software, and real-time applications are explained in chapters 8, 10 and 11 respectively.

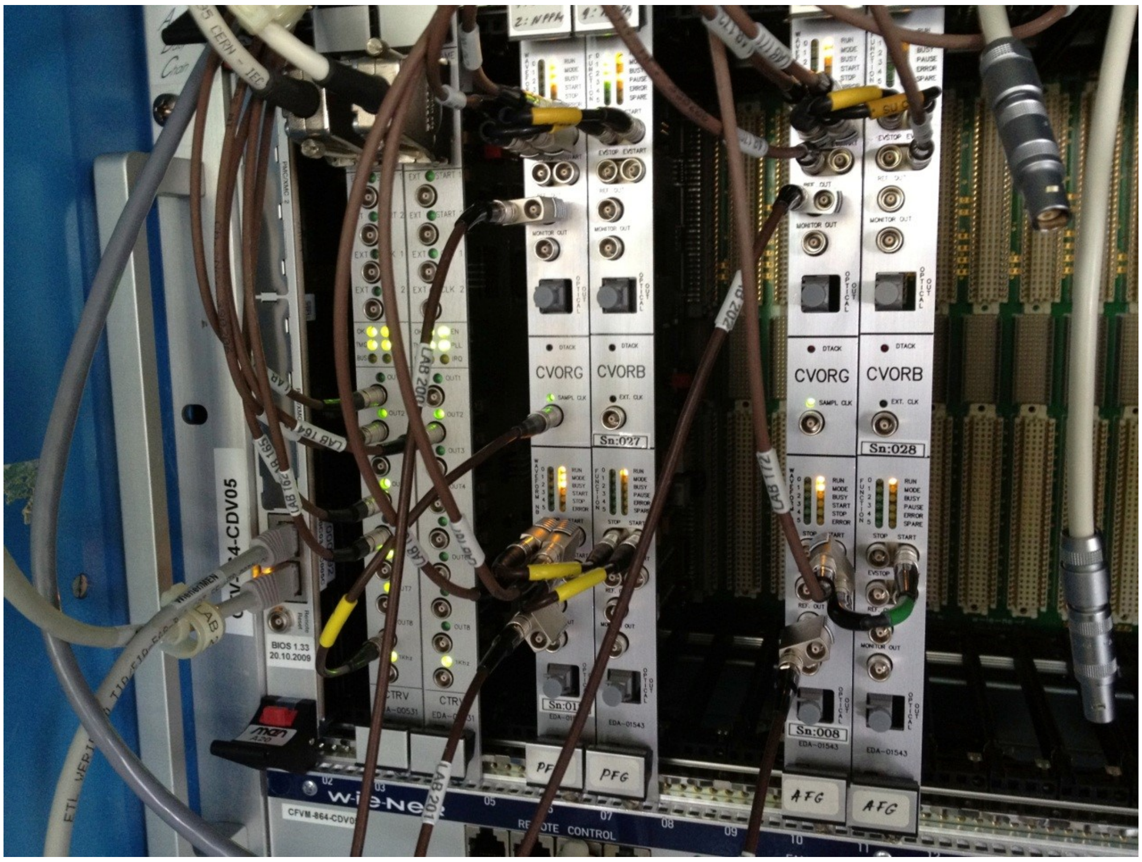


Figure 1.4: Front-End Computer with some electronic modules

1.4 Databases

Usually, databases are considered to be part of the lower/resource tier. However, in the Control System, databases are omnipresent and are somehow connected to all of the layers. The front-end layer takes configuration data from the database, high-level application server and graphical clients are data-driven by the databases. To highlight that fact, as in figure 1.5, we often represent the databases as a vertical layer serving the three tiers directly.

1.5 Remote I/O

It could be argued that the Control System is actually a 4-tier system. In many cases, the interface towards the accelerator hardware is not directly in the front-end chassis but remotely accessed using a fieldbus. So, if the three tiers are based on physical nodes, then the remote Input/Output (I/O) could be considered as a 4th tier: the I/O tier. Nevertheless, to simplify, we consider them to be in the front-end tier and the fieldbus that connects them as an extension of the front-end computer.

Programmable Logic Controllers (PLCs), which have been around for quite a long time, can be considered as remote I/O. More recently, a project to ease the development of remote I/O by providing a communication module and a modern fieldbus was launched by the group [18]. Figure 1.6 depicts the proposed architecture.

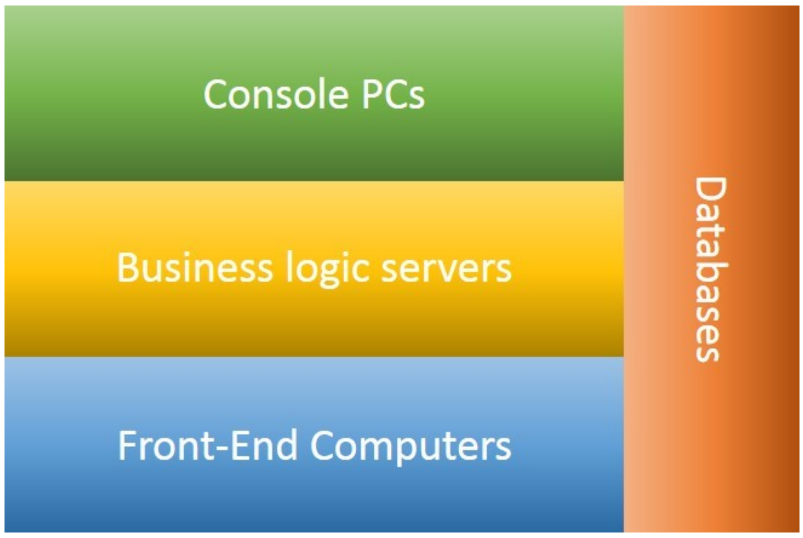


Figure 1.5: The databases are in the resource tier but they interact with all the layers

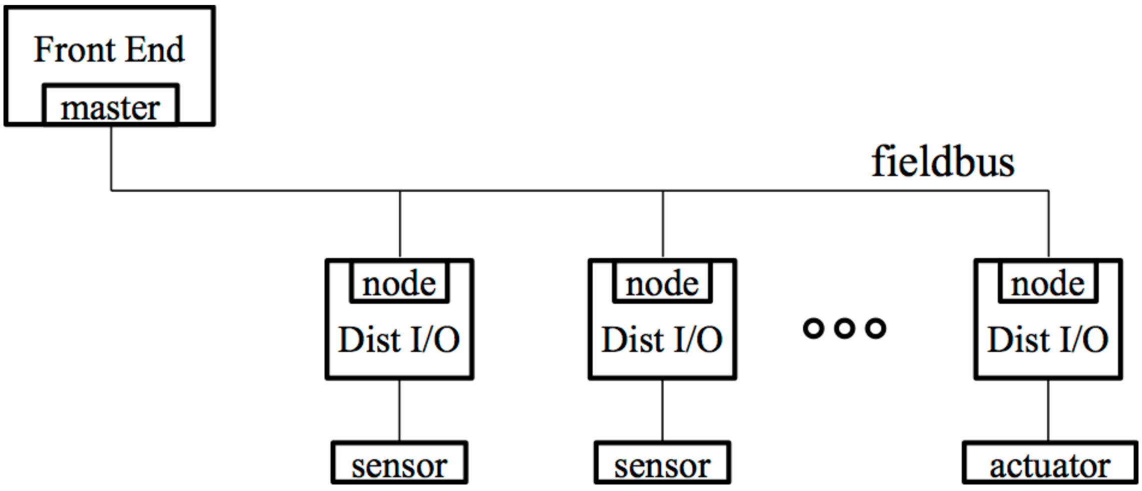


Figure 1.6: Architecture of the remote I/O tier

2. Typical Use Cases

Now that we have an overview of the structure of the Control System's nodes, it's interesting to understand the fundamental data flows and use cases. There are many data exchanges during the accelerators' operational periods but it is sufficient to describe just two fundamental use cases to understand the Control System.

2.1 Control of Setting Values

The control of setting values, which outside the Control System context are commonly known as reference values, is the main use case to control the accelerator hardware. Its data flow is from top (the operator) to bottom (the accelerator hardware). We define a setting parameter as the smallest controllable element in the Control System. Some parameters have a direct hardware correspondence (low-level parameters) while other represent higher-level concepts.

Let's imagine that an operator in the control room wants to change the value of a high-level setting parameter for which there is an algorithm to transform the high-level value into several hardware (low-level) values. Figure 2.1 shows the various components involved in the use case. The steps of the interactions are numbered (Step 1, Step 2, etc.) so that it is easy to follow the explanations on the drawing. One can identify the physical layers described in the previous chapter. In green, we have the Graphical User Interface (GUI) running in the console in the control room. In yellow, there is the back-end server and the database. Finally, the light-blue box represents the front-end computers. There are three front-end computers and, for each, the software and hardware components are represented separately. Figure 2.1 also introduces a new actor, the timing system. The timing system will be described in details in chapters 4 and 24, but for this overview, it is sufficient to know that the timing system issues specific events at specific moments of the beam production cycle. The front-end computers are able to receive and decode these events.

The first step involves the user (e.g. a physicist or an operator) and the GUI. The user wants to set a high-level parameter (PHL1) to a new value x. The user enters x and sends the new value. After some validation of the data, e.g. it does not exceed the maximum value defined for the parameter, the GUI sends the new value to the middle tier process (step 2) using one of the protocols that we support (see part IV for more on the communication protocols and middleware). In step 3, the server processes the change request (aka trim request). The server uses the database to understand how to proceed; are there other parameters related to this one, which algorithm to apply to transform the value... In this example, the high-level parameter is linked to three low-level parameters (PLL1, PLL2, and PLL3). The server computes the new values a, b, and c for the low-level parameters respectively. Before sending the new settings further down, the server stores the new parameter values (x, a, b, and c) into the database (step 4). This allows us to keep track of all the actions taken and to recreate the state of the accelerator at any point in time. The back-end server sends the new values to the front-end computers (step 5). Upon reception, the values are validated by the real-time applications and stored locally. The real-time software must wait for the go-ahead of the timing system as hardware values cannot be changed at any time. At the appropriate moment in the accelerator cycle, the timing system sends an event which is received and decoded by the real-time applications (step 6). The Real-Time (RT) applications write the new values into the hardware (step 7), potentially after an additional transformation of the data to fit the hardware representation. The accelerator hardware now has the new value as decided by the Control System's user.

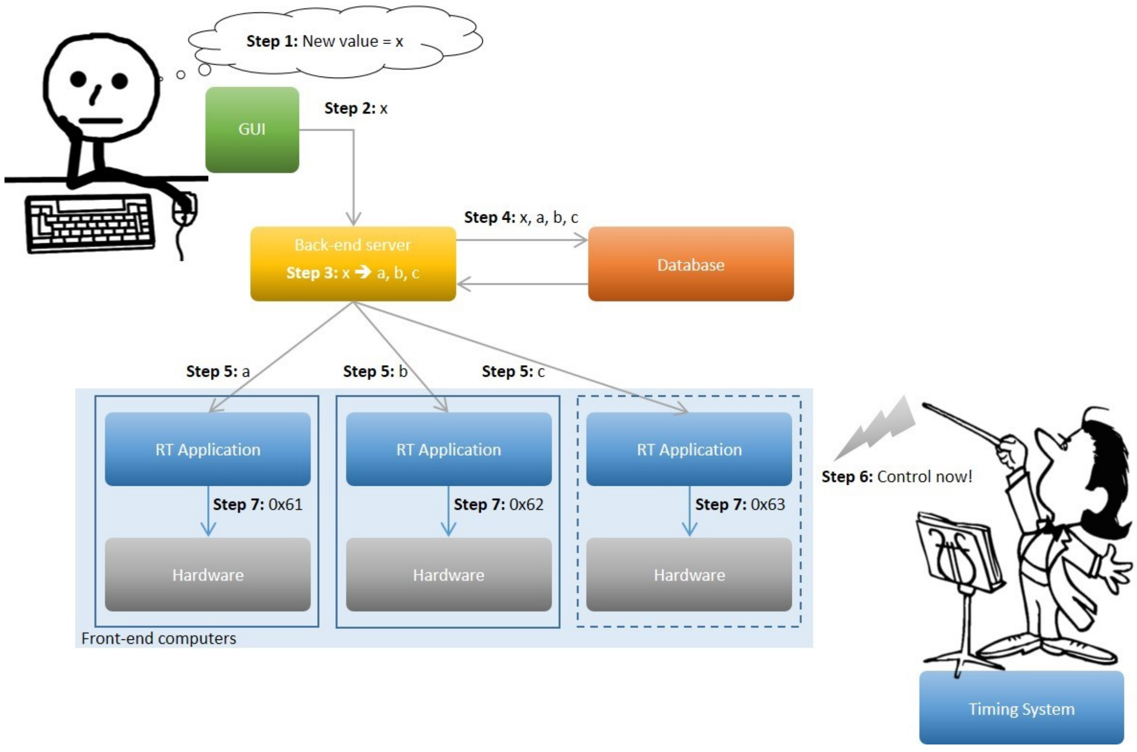


Figure 2.1: Control of setting parameter's value

2.2 Monitoring of Accelerator Variables

While the ability to control the accelerator parameters is primordial, it is actually a use case that is used only a few times per hour. Indeed, once the accelerators are set up and produce beams with the required characteristics, the accelerator settings are only changed slightly for further optimisation or during accelerator study sessions (aka Machine Development (MD)). On the other hand, the monitoring of the accelerators' components is performed continuously in cycles, less than once every second in some cases. In addition to its repetition rate, monitoring also produces much more data.

Before going into details, we should differentiate between on-line monitoring and off-line monitoring. The first is more like streaming values directly to the end user while the second decouples the acquisition and analysis part. However, we can look at both scenarios together, as only the last steps are different.

Acquisition value streaming

In the use case depicted in figure 2.2, the user receives a continuous stream of acquired data. This allows him to ensure that the accelerator performs as expected and gives quick feedback after having changed a control value.

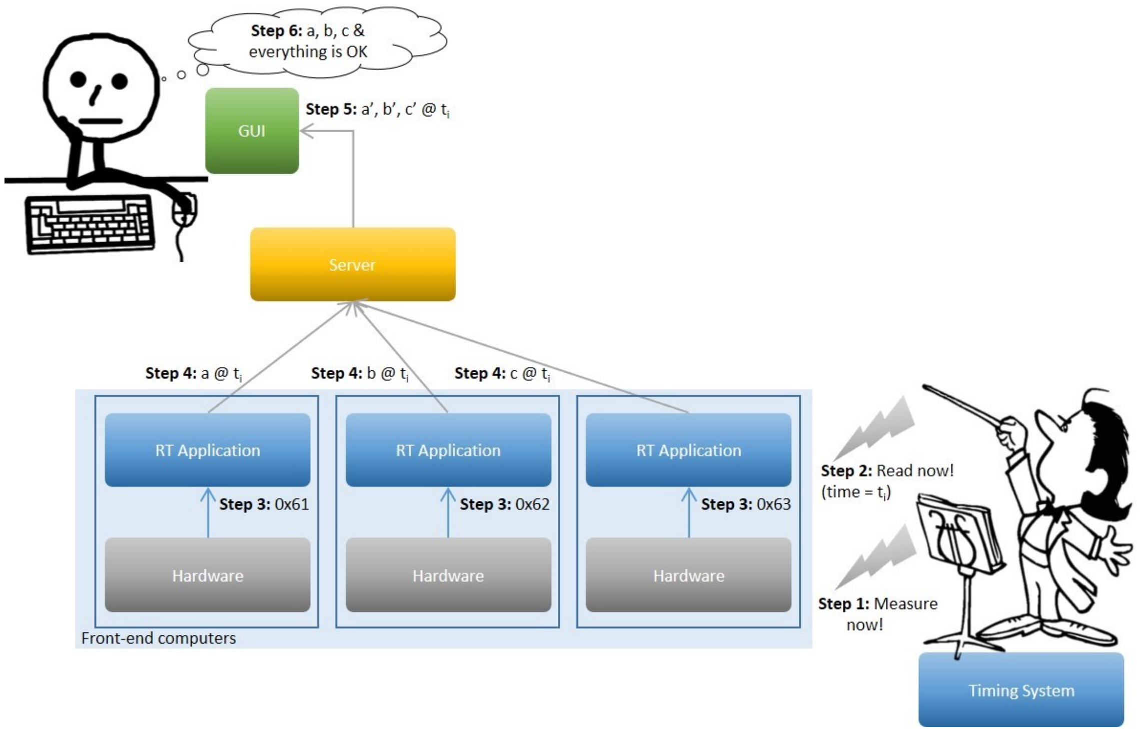


Figure 2.2: Acquisition value streaming use case

The timing system is continuously delivering events that are used to trigger the acquisitions of the various accelerator elements (step 1). On top of that, the timing system also delivers events that are intercepted by the real-time applications and instructs them to read the acquired values (step 2). Note that, like all timing events, the "Read now" event has a timestamp that is very important to allow the correlation of data originating from different sources at different locations. Next (step 3), the real-time applications access the hardware and read the low-level values. After some conversions and association with the event timestamp, the RT applications publish the updated acquisitions over the network (step 4). The server receives the data from the different sources and uses the timestamps to group the data belonging to the same acquisition cycle into a single set. Before sending the data to the GUI (step 5), some post-processing is done. A wide range of post-processing exists, from single value comparison to more advanced computation to create high-level acquisition parameters, similar to those described in the control use case. In this example, the values are simply enhanced with status information, allowing the user (step 6) to ascertain that the system is behaving as expected.

Acquisition value logging

Given the large amount of data produced, it is clearly impossible to manually scrutinise every value for every cycle. On the other hand, it is paramount that we can analyse past events and look at trends. This is why, in addition to acquisition value streaming, we also perform acquisition value logging. The first four steps of this use case are exactly the same as the previous use case. The main difference is that the acquired values are not post-processed and streamed directly to the user, but instead, they are stored in a database. This is depicted by the step 5 in figure 2.3 where one can see the values and the time at which they were acquired are sent to the database. The steps 1 to 5 are then repeated continuously, without user intervention, based on the rhythm given by the timing system.

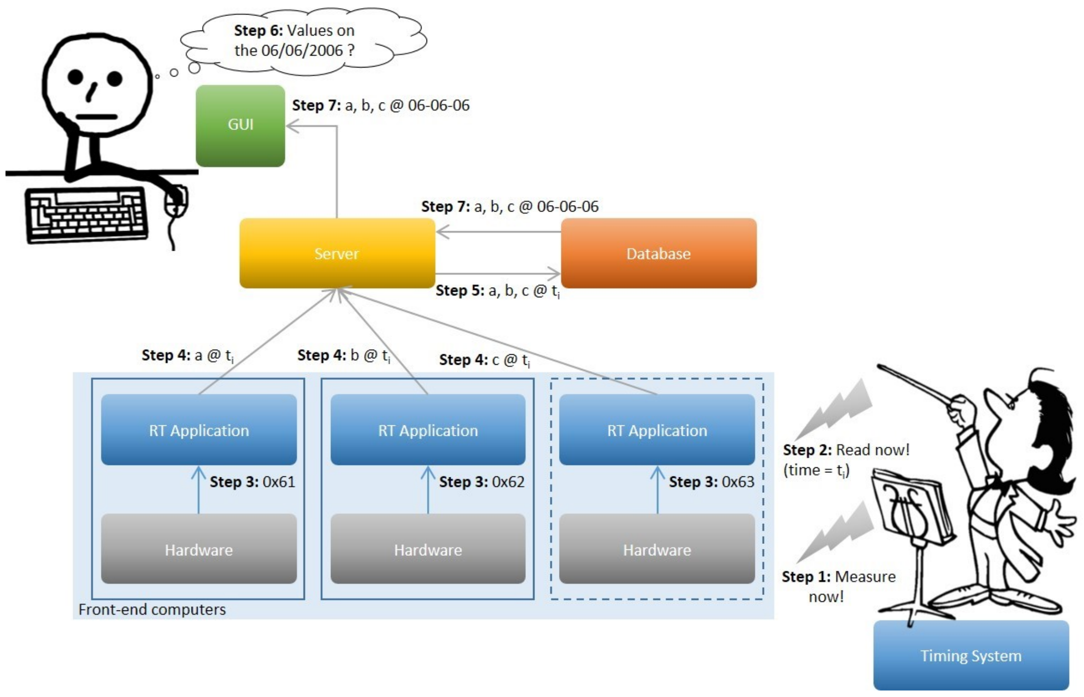
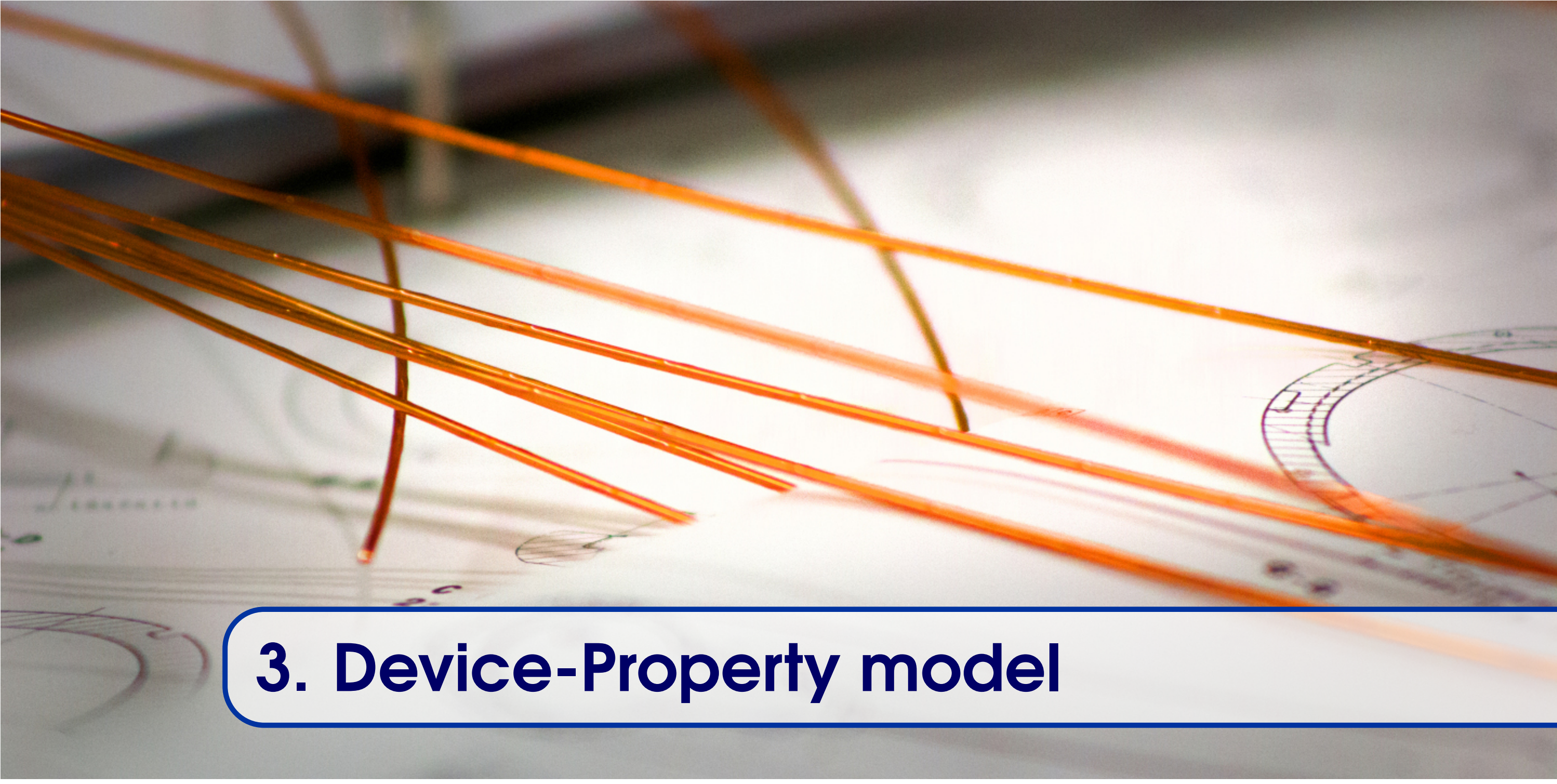


Figure 2.3: Acquisition value logging use case

Even up to 20 years later, a user can query values from a given date at which an acquisition was done. In our example (step 6), the user enters the variables to retrieve along with the date and the GUI sends the request to the middle-tier server. The server retrieves the requested data from the database (step 7) and sends it to the GUI for further analysis by the user (step 8).



3. Device-Property model

The device-property model defines the structure, as well as the operations and their behaviours, required for the exchange of data between the low-level software and the high-level software. The two fundamental concepts are the *device* and the *property*. It is a straight-forward object-oriented model: each piece of equipment is a device, for example, a power converter is a device, a beam current transformer is a device, etc. A device has properties and one can read (*get* operation), write (*set* operation) or monitor (*subscribe* operation) a property. As in object-oriented languages, objects are instances of classes and therefore devices are instances of classes as well. It is at the class-level that properties are defined with their content and the operations they support. As a result, all devices of a given class have the same properties and behaviour. A property contains one or several value-items[1], much like a C structure contains attributes. Each value-item has a basic type (double, int, char, etc.) that can be a scalar, an array or a 2D array. Figure 3.1 represents a simplified class diagram of the device-property model. The device-property model has been used for more than 20 years and has evolved over time and continues to do so. For example, the early versions did not have the value-item concept and properties held a single value. The device-property model described below is based on the FESA3 V4.0 meta-model. Other implementations may vary slightly.

There are three types of properties (setting, acquisition, and command) and each type limits the operations available. The main usage of setting properties is to allow upper layers to send hardware settings to the FEC layer. The typical usage of an acquisition property is to send values acquired by the hardware to the high-level component. The command property, as its name indicates, is to model commands that are given to the FEC software. The setting property can be read and written. The value-items of a setting

[1] In the high-level components and libraries, a value-item is called a parameter or a field. As this chapter is dedicated to low-level software, we decided to use the low-level software term value-item.

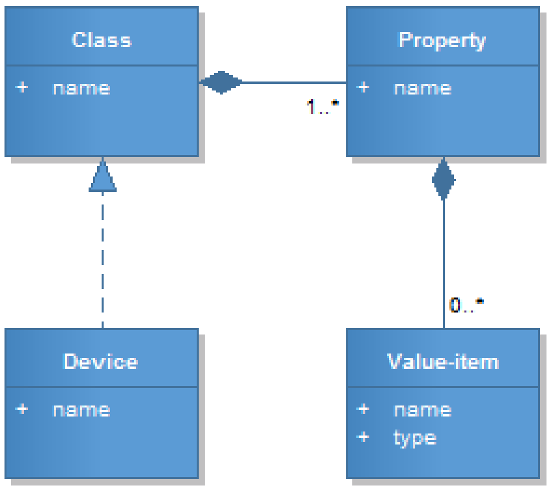


Figure 3.1: Device-property model class diagram

property can all be read but some of them may not be writable. This is typically the case for value-items providing additional information about another value-item of the property. For example, the value-item voltage, representing the voltage produced by a power converter, has a sibling value-item `voltage_max` representing the maximum voltage allowed. The first item can be written but the second cannot. An acquisition property can only be read and therefore all its value-items are read-only. Similarly, as a command property can only be written, all its value-items are write-only. In addition, since it is possible to have commands without any parameters, a command property can be defined without value-items. Figure 3.2 summarises the different property types.

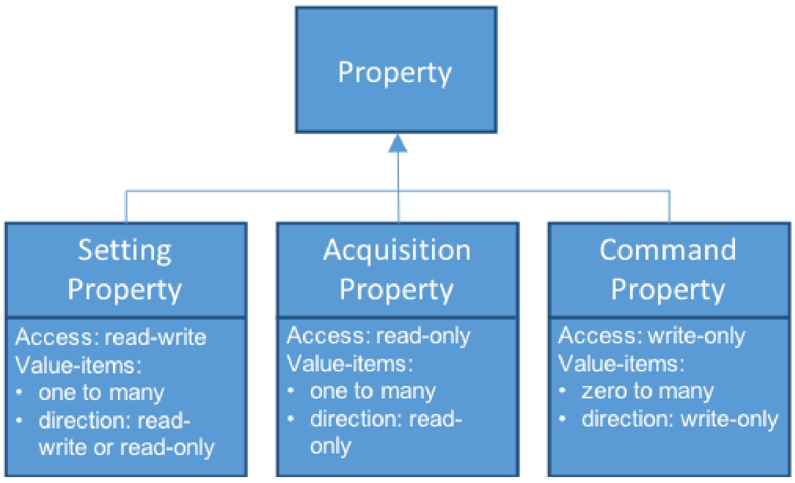


Figure 3.2: Types of properties

The readable properties can also be monitored (or subscribed to). When somebody subscribes to a property, they register their interest in receiving updates whenever the property is refreshed. By convention, the rate at which the property is notified and updates are sent depends on the property type but also on the device's attributes. Setting properties are normally notified only after a set operation, as their values do not change otherwise.

Acquisition properties can be notified after every accelerator cycle or only whenever something interesting happens. When a subscription is created, the FEC software is expected to immediately send the current value of the subscribed property. This is called the first update and is used to initialise the subscriber. For multiplexed properties, several first updates, one per timing user, are sent to the subscriber. The subscriber can specify a selector in addition to the device/property to be monitored. This selector is used to filter the updates and only send those that are of interest.

In order to be able to control the hardware following the instructions of the timing system, the FEC software needs to have access to all of the settings for all of the timing users. However, even if the accelerator is time-multiplexed, not all settings are; the properties and the devices can be defined as multiplexed or not. When accessing a multiplexed property, the client must specify a selector, similar to the one given for a subscription. It is important to note the difference between selectors for set/get and subscription operations. For the subscriptions, the selector is only used for filtering updates i.e. choosing the updates to be sent to a particular client. However, for the set/get calls, it is used to de-multiplex the values. As mentioned in section 4.1, the timing user is employed for the value multiplexing and, therefore, only selectors based on the timing user can be provided in set/get calls. The same constraint applies to subscriptions to setting properties, as they are only updated on a set and, therefore, the update filtering is not possible on other timing fields.

4. Timing

CERN's accelerator complex is comprised of 12 different machines [2], as shown in figure 4.1. The beam's energy increases as it is transferred from one accelerator to the next. The system to control the accelerator equipment is highly distributed. In order to synchronise the accelerators and optimise the performance of the whole complex, a sophisticated timing system is required.

We can separate the timing system into two levels; the beam scheduling and the event and data distribution. The beam scheduling, or sequencing, is about deciding which beam to produce in a given accelerator at a given time. The event and data distribution is about sending the right events at the right moment to the equipment, along with data related to the cycles being produced.

In this chapter, we focus on the timing concepts that are required to understand how the components of the Control System work and interact with each other. The implementation details are given in chapter 24 and, for further reference, the reader can consult [40].

4.1 Sequencing

The Large Hadron Collider (LHC) experiments are the main focus for physics at CERN, but there are also many other particle beam users. In addition to the next accelerator in the chain, almost all accelerators also have one or more direct clients. For example, the PS Booster (PSB) provides beams to ISOLDE, the Proton Synchrotron (PS) provides beams to the Anti-proton Decelerator (AD), the East Area and the Neutron Time-Of-Flight facility (nTOF) experiment, etc. Furthermore, the time required to produce a beam can vary significantly from one accelerator to the next, and while the accelerator N is producing

[2]From 2017, CTF3 was converted into a new facility called CLEAR.

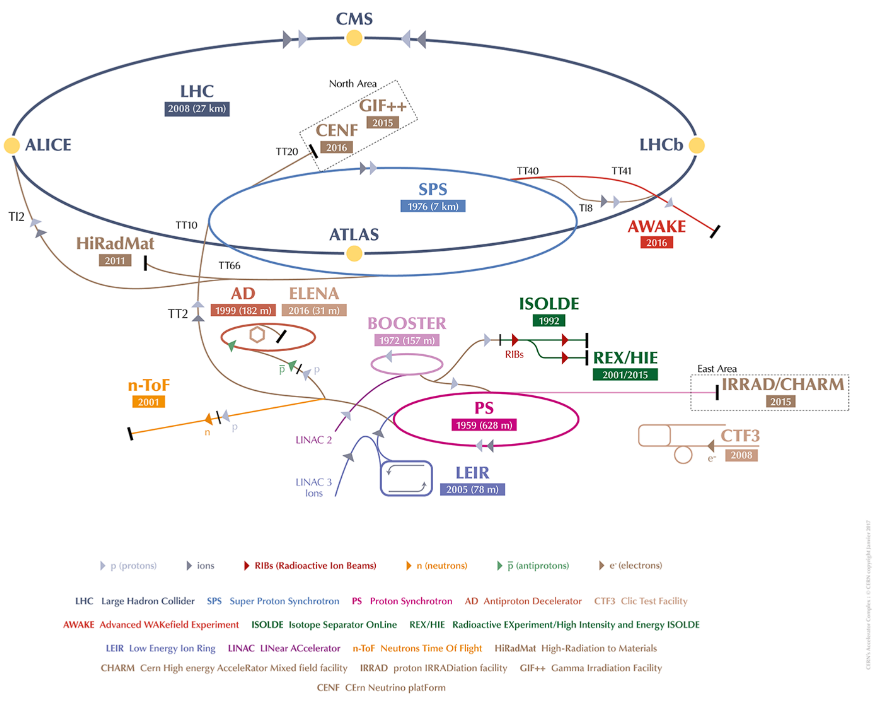

CMS
LHC
2008 (27 km)
North Area
GIF++
2015
CENF
2016
ALICE
LHCb
TT20
TT40
TT41
SPS
1976 (7 km)
TI8
TI2
TT10
ATLAS
AWAKE
2016
HiRadMat
2011
TT66
AD
1999 (182 m)
ELENA
2016 (31 m)
TT2
BOOSTER
1972 (157 m)
ISOLDE
1992
RIBs
REX/HIE
2001/2015
East Area
IRRAD/CHARM
2015
n-ToF
2001
PS
1959 (628 m)
CTF3
2008
LINAC 2
LINAC 3
Ions
LEIR
2005 (78 m)
p (protons)
ions
RIBs (Radioactive Ion Beams)
n (neutrons)
p̄ (antiprotons)
e⁻ (electrons)
LHC Large Hadron Collider
SPS Super Proton Synchrotron
PS Proton Synchrotron
AD Antiproton Decelerator
CTF3 Clic Test Facility
AWAKE Advanced WAKefield Experiment
ISOLDE Isotope Separator OnLine
REX/HIE Radioactive EXperiment/High Intensity and Energy ISOLDE
LEIR Low Energy Ion Ring
LINAC LINear ACcelerator
n-ToF Neutrons Time Of Flight
HiRadMat High-Radiation to Materials
CHARM Cern High energy AcceleRator Mixed field facility
IRRAD proton IRRADiation facility
GIF++ Gamma Irradiation Facility
CENF CErn Neutrino platForm

Figure 4.1: CERN's accelerator complex

its beam, the accelerator N-1, which provided the beam, is potentially inactive, as depicted in figure 4.2.

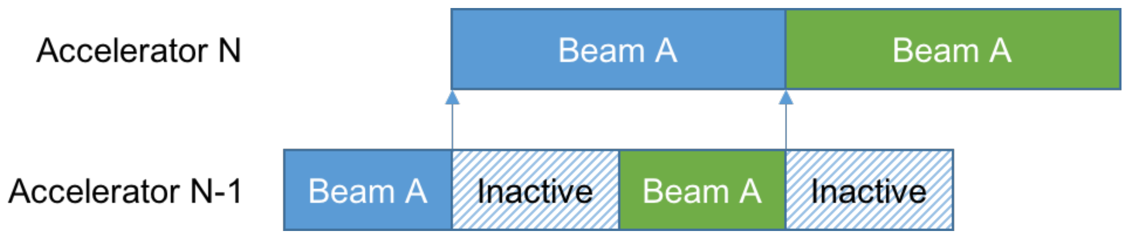


Figure 4.2: Unoptimised usage of the accelerators

Due to the requirements of handling multiple clients per accelerator and optimising the duty cycle, an advanced timing system has been put in place. The timing system allows accelerator operators to schedule different beams in the different accelerators and optimise the particle throughput. As depicted in figure 4.3, the accelerators continuously switch between producing different types of beams and as a consequence, the Control System has to dynamically re-programme the hardware. This behaviour is commonly referred to as time-multiplexing but the term Pulse-to-Pulse Modulation (PPM) is also often used at CERN.

The main group of machines forms the LHC Injector Chain (LIC). The LIC is made of the LINAC2, PSB, PS and SPS for the proton chain and the LINAC3, LEIR, PS and SPS for

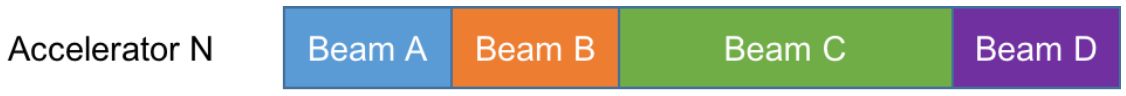


Figure 4.3: Accelerators are time-multiplexed

the ion chain. The other accelerators such as AD and ELENA are independent and are only coupled to the LIC for beam transfers [24]. The LHC is also separate, from a timing point of view since, as a collider, it works with fills rather than cycles. An LHC fill is made of a filling phase, an acceleration phase, and a collision phase, which can last several hours. In the remainder of this chapter, we focus on LIC scheduling as it is the most sophisticated.

The operators in the control room create and send the beam sequence for all the LIC accelerators to the central timing. The sequence is continuously repeated until it is changed. At runtime, the central timing collects information from external sources, such as magnet interlocks, beam requests, etc. to decide whether the programmed beams can be executed. The behaviour of the central timing is described by equations (aka FIDO equations), which evaluate the external conditions according to internal logic. For each beam, each accelerator executes one or several cycles in order to take the beam from the upstream machine, accelerate it, and deliver it to the next one in the chain. In the Control System, a cycle is represented as a timing user. A timing user has a length, which is always a multiple of a 1.2-second basic period. Figure 4.4 represents an LHC 25-nanosecond beam, as scheduled in the PSB, PS, and SPS. Note how the PSB and the PS execute several cycles for a single SPS cycle. In this example, the PSB cycles take 1 basic period (1.2 second), the PS cycles take 3 basic periods (3.6 seconds) and the SPS cycle takes 11 basic periods[3] (13.2 seconds). Figure 4.5 represents the same beam from a controls perspective with the correct timing user names for the different cycles.

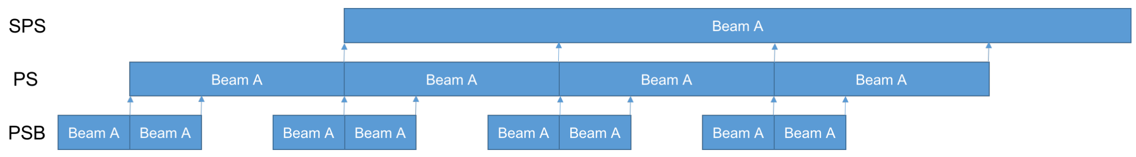


Figure 4.4: The LHC 25ns beam in the PSB, PS, and SPS

The concept of a timing user, or simply a user, is very important and present in most of the Control System's components. All of the accelerator multiplexing is done using the user as a demultiplexing key. Even though the final goal is to produce beams and the central timing schedules beams, the Control System acts on users. For a given beam sequence, the set of users in a given accelerator is called the supercycle, as it is a cycle of cycles.

Creating the accelerator complex's beam schedule is not an easy task, as the beam structure can be quite complicated. Fortunately, the operators have a graphical tool to help them specify what they want to execute in a Beam Coordination Diagram (BCD).

Whenever the conditions to produce a given beam are not met, the central timing skips it. To avoid wasting time, the operator can specify spare beams (i.e. a spare user in each accelerator) that can be produced instead, as depicted in figure 4.6.

[3]The actual LHC25NS user in the SPS is longer

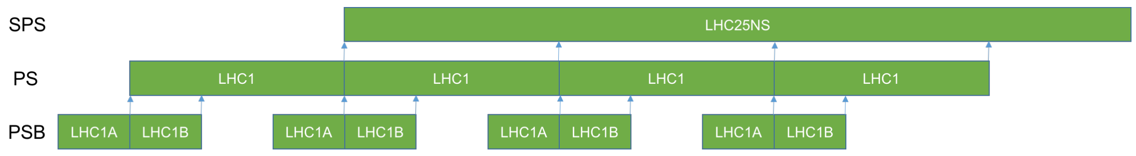


Figure 4.5: Timing users producing the LHC 25ns beam

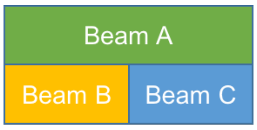


Figure 4.6: Normal and spare cycles

The rules of execution for the spare beams are identical to those of the normal beams. When a spare beam cannot be played either, the central timing schedules it, but without particles.

There are two main situations where the normal/spare mechanism is very useful. The first one is when the beam needs to be setup or improved, but its next destination is not available. In this case, the spare beam is the same as the normal one, but the destination is set to a beam dump. The second use case of normal/spare is when we can serve more particles to a destination by replacing a single multi-user beam by several smaller beams. This happens when a single 2.4-second cycle for both the east area and the nTOF experiment is replaced by two 1.2-second cycles exclusively for nTOF whenever the east area does not request or cannot take the beam.

4.2 Events and Data

Once the central timing has decided what will be produced next, it communicates this information to the rest of the Control System using events and a data stream called the telegram.

Typical timing events are the main events in a particle accelerator's cycle such as start-cycle, injection, ejection, etc. Events in the central timing can be linked to one another and it is easy to create groups of events all related to a parent event (the virtual event). For example, the master-injection event is linked to other events such as the forewarning-injection, 900 milliseconds before injection, and the warnings 10 and 20 milliseconds before injection. Changing the injection time will automatically change the other events. The Central Timing events (CTIM) are distributed on the General Machine Timing (GMT) network and received by the timing receivers installed all around the accelerator complex. Local-event software, Local Timing events (LTIM), runs on the timing receivers and produces bus interrupts and electrical pulses. Thanks to the combination of generic hardware and highly configurable software, the distributed timing system is very flexible and many different schemes can be implemented, from the simple repetition of a CTIM to complex pulse-burst generation with RF clock resynchronisation.

There are several pieces of information attached to a timing user. The most important one is its name but the telegram also contains its length, its destination, its position in the super-cycle, and other low-level timing-specific values. Each piece of information is transported in a telegram group as a 16-bit value due to the technical constraints of the GMT network. The old TeleGraM (TGM) library exposes the raw values to the clients without any formatting and therefore is not user-friendly. To improve the situation, the latest timing library (TimDT) hides the telegram groups and instead exposes the timing fields that have already post-processed values. For example, the LHC energy that is available as a telegram group with the encoding "1 bit equals to 120 GeV" is directly available as a timing field with its value in GeV. The current situation is not yet homogeneous and the low-level Control System (Front-End Software Architecture (FESA), etc.) already works with the new timing fields, but most of the high-level Control System is still based on the telegram group.

4.3 Cycle Selector

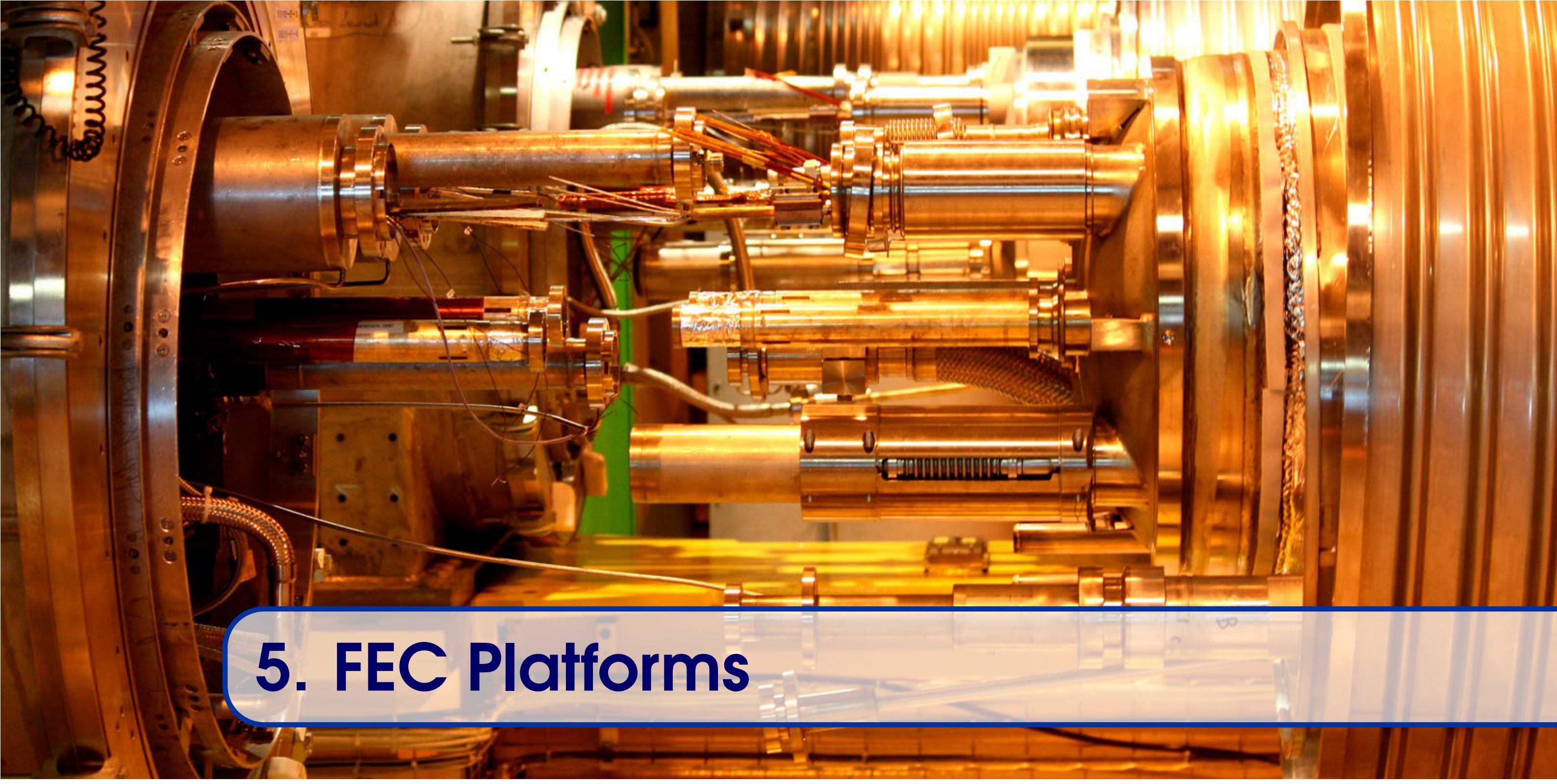
As explained in the previous section, the concept of the timing user is omnipresent in the Control System. Therefore, one needs a common way to indicate the timing user one wants to use in the various Application Programming Interface (API)s. A timing domain defines a timing environment with one or several accelerators. A supercycle is attached to the timing domain rather than the accelerators; in other words, one can schedule cycles (one at the time) in a timing domain. While most accelerators have their own independent timing domain, the small linacs (LINAC2, LINAC3, and LINAC4 are part of the PSB (LINAC2 and LINAC4) and LEIR (LINAC3) timing domains. It is important to not confuse the accelerator name and the timing domain. It is a nuance that often leads to confusion especially since some accelerators and some timing domains share the same name (e.g. SPS).

Every timing domain has several telegram groups, as explained in section 4.2, and every group has a unique name in its domain. The most common group is USER and contains the name of the timing user. Depending on the domain, one has groups such as DEST for destination, PARTY for particle type, or ENG for the current LHC energy.

A cycle selector, often known as a selector, is a triplet separated by dots such as XXX.YYY.ZZZ. The first part is the timing domain's name such as PSB, CPS, LHC. The second element is the telegram group's name, for example USER or DEST. The last part is the value of the telegram group one is interested in. If one wants to access devices in the PS accelerator for the user LHC1, one must write CPS.USER.LHC1. CPS is the timing domain the PS accelerator is in and one wants the LHC1 user. As mentioned earlier in the chapter, the accelerator multiplexing is done with the user as a key, therefore, most operations (get, set) can only be done with USER-type selectors. Nevertheless, in the case of subscription, one can use another telegram group to select one or several cycles. For example, CPS.DEST.SPS selects all the users with the SPS as destination.

II

Controls Hardware



5. FEC Platforms

The Front-End Computers' (FEC) main mission is the low-level control and acquisition of data from the accelerator hardware. A FEC is a rack-mountable electronics enclosure, built to survive in an industrial environment. Generally, it is comprised of a chassis, a backplane, a power supply and in some cases a fan tray. Its modular design allows the use of electronic extension boards, which are plugged into the backplane. These boards interface with the accelerators' equipment to provide control and acquisition through electrical signals. The main board is a diskless Single Board Computer (SBC) that has all the usual components that can be found in a desktop computer: a CPU, some Random Access Memory (RAM), bus controllers and bridges towards other buses (e.g. North PCI bridge, PCI to VME bus...). The SBC is plugged into a designated slot in the backplane to control the other extension boards. FECs conform to one of several bus standards that are chosen depending on an application's specific technical requirements (e.g. the data rate, cost) and the availability of ready-to-use modules. Over the years, the controls group has added support for several standards and we regularly look at the market evolution in order to offer an optimised range of solutions. The latest front-end platform survey is available on the BE-CO wiki space [4]. In addition to the different bus standards, the FECs can be categorised as either open or closed enclosures. Figure 5.1 depicts the FEC platforms' family tree indicating the standards currently supported or used in the controls group, as well as how the standards relate to each other.

5.1 Open Enclosures

Open enclosures allow easier access to extension boards than closed enclosures, which require disassembly in order to change modules. A typical open-enclosure is a metal chassis with the front face removed through which the extension boards are inserted and

[4] `https://wikis.cern.ch/display/HT/Future+Front+End+Platforms`

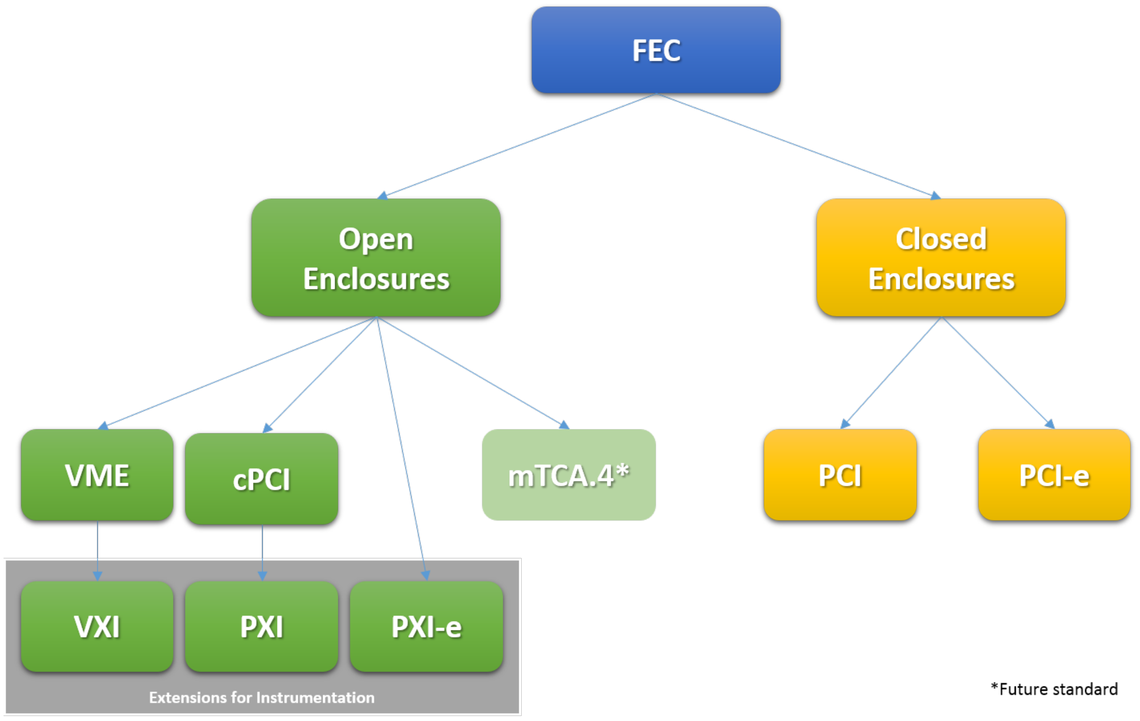


Figure 5.1: Family tree of front-end platforms

clipped into place. Modules come in either 3U or 6U formats. The crates themselves can be 8U in height (6U + 2U fan tray) with 17 slots for modules, or 4U or even 2U (which position the extension boards horizontally) with 4 slots and 2 slots respectively. The power supply is located at the back and the fan trays have an Ethernet connection to enable remote diagnostics. Connections to external equipment can be done either directly on the front-panel of the extension board or at the back of the chassis using a Rear-Transition Module (RTM).

As with any electronic crate, open enclosures come with their challenges. Electronic boards dissipate heat and need cooling, produce electro-magnetic noise and need shielding, and are plugged into the chassis and need firm fixations. In order to ensure the optimal air flow through the crate and to limit electro-magnetic noise coming in and out, empty slots should be closed with a front panel. Figure 1.4 shows an example of an open enclosure without the empty slot covered. The advantage of easy to install modules can also be a potential weakness. Modules can suffer from bad or deteriorating contacts caused by vibrations and dust etc. Therefore, it is necessary to screw the modules in, to bind them firmly to the chassis.

5.2 Closed Enclosures

Open enclosures are generally more suited to our environment but are expensive. Some applications have a lot of installations with a common configuration and only require a few extension boards. For example, the LHC World Factory Instrumentation Protocol (WorldFIP) gateways have only one timing receiver and up to two WorldFIP bus controllers. In these situations, it does not make sense to install full VME crates as there are more cost-effective solutions available.

Following a market survey, the controls group decided to support the PICMG 1.3 closed enclosures, which are System Host Board (SHB) based on PCI-SIG standards (PCI, PCI-express). In order to facilitate the installation and maintenance, pre-mounted systems are always kept in stock.

In practice, ready-to-use chassis can be problematic. If a fault occurs, the diagnostic is harder, as a complete exchange is not always desirable. Removing all the cables and the chassis can be time consuming and therefore one wants to ensure the faulty element is the chassis before exchanging it. To facilitate the diagnostics and maintenance, some chassis are mounted on rails and fitted with longer cables. This solution increases the overall cost and is therefore normally reserved for laboratories, where equipment is changed frequently and access to the boards is more important.

Closed enclosures can also suffer from heat dissipation problems, as discovered when using fast digitiser modules, which typically draw a lot of current and therefore produce a lot of heat. Unlike the open enclosures, which push the air flow between the extension modules, the PICMG 1.3 chassis attempt to extract the hot air without controlling the air flow. In order to solve the heat problem, more powerful fans had to be installed and modules had to be ordered with additional on-board fans.

5.3 Backplanes and Buses

As described above and depicted in figure 5.1, the chassis can have backplanes using different standards. The next chapters describe the standards supported by BE-CO.

5.3.1 VMEbus

Versa Module Europa (VME) is the oldest bus standard supported by BE-CO. It is an asynchronous parallel bus, originally developed for the Motorola 68000 line of CPUs. VITA manages the different versions of the VME standards (`http://www.vita.com`). The success of the standard stems mainly from the fact that many Commercial-Off-The-Shelf (COTS) modules are available and that interoperability is guaranteed by the standard, which means that any VME crate can be used with any VME module.

VME has a separate 32-bit data bus and 32-bit address bus, plus a few control lines to manage the protocol. In addition to the read and write operations, VME supports modules requesting the attention of the CPU through interrupts.

VME is the least sophisticated protocol supported by BE-CO, and it is possible to describe the read cycle in simple terms (figure 5.2).

In the first step, the CPU puts the address on the address bus from which it wants to read. The address corresponds to one memory space in one, and only one, module in the crate. As this is a read cycle, the Write (WRITE) and Interrupt Acknowledge (IACK) lines are negated. SinceVME is an asynchronous bus, the CPU asserts the address strobe line, to indicate to the modules that the address is now available for reading. The CPU asserts one or both data strobes to indicate where on the data bus it will expect to read data. As soon as the slave is ready, it puts the data on the data bus and asserts the data transfer acknowledge

(DTACK) line. Once the CPU has read the data it negates the strobes to indicate that it has read the data. In turn, the module will negate DTACK to indicate the end of the cycle. If at any moment, something goes wrong, the bus error line is asserted and the cycle is aborted.

Typical read-write access on a VME system is approximately 1us per register. For cases where higher throughput is required, the Block Memory Access (BMA) is available to transfer a block of memory from the VME module to the Single Board Computer (SBC). BMA can work in combination with Direct Memory Access (DMA) allowing us to transfer data from a VME boards to the RAM without using the CPU.

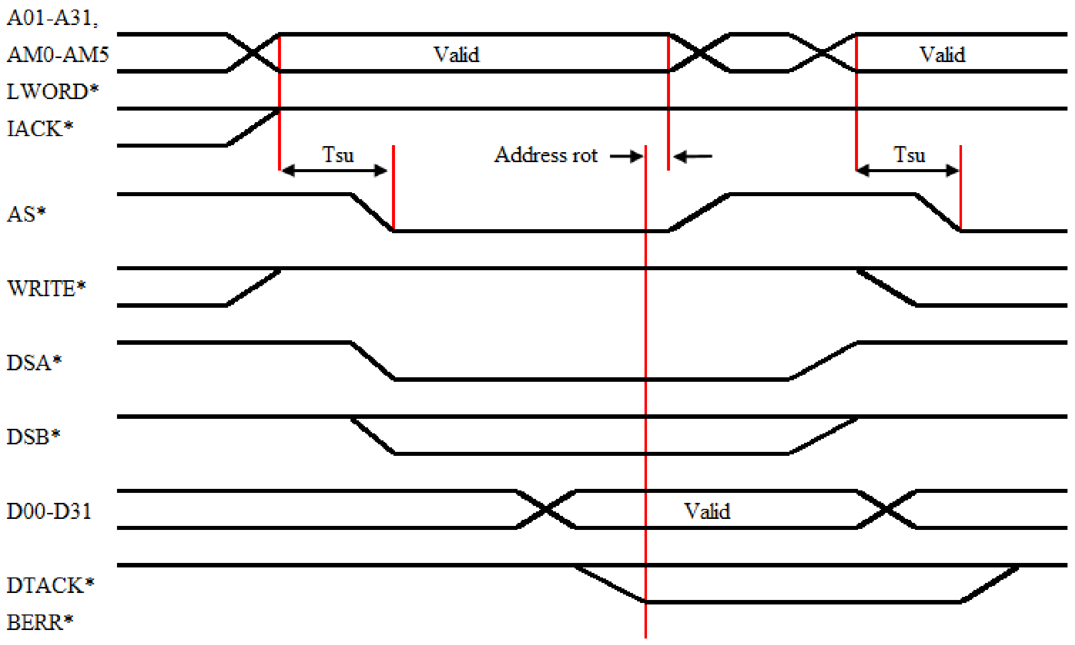


Figure 5.2: VME read cycle

64-bit Evolution (VME64x)

One of the major evolutions of the VMEbus was its extension to a 64-bit address and data bus. In addition, this new version also provides software configuration and geographical addressing, reducing the configuration required when installing a new module. Prior to this, it was necessary to set the base address, interrupt number, address modifier etc. using physical jumpers on the modules. With VME64x, the configuration of the module is done through software and the addressing of the configuration space is derived from the slot number of the module in the crate. This feature obviously makes maintenance easier but it removes the possibility of replacing a module by installing another module in another slot.

Other evolutions of the VME standard, including VME Switched Serial (VXS) and VME PCI eXtension (VPX), are sometimes used in niche applications in other groups e.g. the BE department's Radio Frequency (RF) group. Even though our VME64x SBCs are used in these crates, these platforms are not supported by the controls group.

VME Single Board Computers

The first Single Board Computers (SBC) used in the VME crates were based on the Motorola 68k family running the hard real-time LynxOS operating system. The following generations were based on PowerPC CPUs, still running LynxOS. The latest generation, the MEN A20, represents a 180-degree turn for the group as we decided to move to an Intel CPU (Intel Core 2 Duo for the A20) and to drop LynxOS support and migrate to Scientific Linux CERN; a version of Linux with low-latency patches. Modern Intel CPUs provide a multi-lane PCI Express (PCIe) bus to interface with the external world and, in order to connect the CPU to the VME, a bridge is required to translate the VME protocol to PCIe. For the MEN A20, a COTS bridge, the TSI148 chip, is used. The next generation of VME SBCs will show yet another increase of CPU power and memory space as the MEN A25 has 4 cores (8 threads) and at least 8 GB of RAM. For the MEN A25, it was decided to use an Field-Programmable Gate Array (FPGA)-based open-source design for the VME bridge to avoid the recurring problem of chip obsolescence [17]. Compared to the RIO2 PowerPC available at the end of the 90s (32-bit single core clocked at 75 MHz with 16 MB RAM), the front-end computers now have many more capabilities than when the system was designed.

5.3.2 VXI

Based on the VMEbus, VME eXtension for Instrumentation (VXI) defines additional bus lines for timing and triggering, but also protocols for configuration; something that was missing from the original version of VME. The standard specifies different mechanical requirements with modules that are longer and wider than the usual VMEs. VXI chassis can only host a maximum of 13 modules, which is significantly less than our biggest 21-slot VME chassis.

The only system that has ever used VXI in the BE-CO Control System is Open Analogue Signal Information System (OASIS) (see chapters 11 and 23 for more details). Even though the VXI chassis used in OASIS were for analogue signal digitalisation, we never actually used the timing and trigger lines. On the other hand, we benefited from the geographical addressing for configuration. VXI is now deprecated and the current plan is to renovate all of the VXI-based installations during Long Shutdown 2 (LS2), therefore ending the BE-CO's support of the platform.

5.3.3 PCI and Compact PCI

Created in the early 90s, the Peripheral Component Interconnect (PCI) local bus standard defines a synchronous (single clock) parallel bus. The controls group started supporting PCI, first with Compact Peripheral Component Interconnect (CompactPCI) and later plain PCI in the early 2000s, as we were facing two challenges with the VMEbus. The first reason was bandwidth limitation and unavailability of COTS modules for fast signal digitalisation. The second issue was the platform cost, as explained in section 5.2. As PCI was originally associated with Intel based computers, the adoption of PCI also marked the introduction of the Intel CPUs in the Control System, which was PowerPC-driven at the time.

There are several variants of the PCI bus and the PCI-SIG consortium specifies all PCI related standards. Like the VME bus, interoperability is guaranteed by the standards. A

PCI bus can be 32-bit or 64-bit and is synchronous to a single bus clock at either 33 MHz or 66 MHz. With these clocks and word sizes, the PCI bus can transfer data from 133 MB/s to up to 533 MB/s (a normal VME64x access will have an 8 MB/s throughput). Similar to the VME bus, the PCI bus supports slaves requesting the master's attention through interrupts but, unlikeVME, the interrupt lines are configured automatically, making the installation and maintenance simpler. Another key difference with respect to the VME is that the address bus and data bus lines are shared, meaning that during a read/write cycle the lines will first be address lines and then data lines. This makes the basic operations more complex as a state machine is now required. A simplified description, such as the one presented for the VME bus, is clearly outside the scope of this introductory document.

One interesting feature of the PCI bus is that it uses reflected-wave switching, which impacts the maximum size of a PCI bus. When the backplane is longer, such as in the CompactPCI chassis, PCI bridges chaining several buses have to be installed in the backplane. The modules in the chassis will then be on different buses. Fortunately, this chaining is transparent for the user, who only requires a bus and a slot number. In addition, while there are no jumpers to configure with the PCI, as we had to with the VME, a PCI chassis does not have a standardised correspondence between a logical and a physical location. Each module is inserted into a slot on one of the buses and the relationship between the physical slot number and the pair PCI bus/slot depends not only on the backplane but also the CPU type. As we want the software to be configured based on physical slot, we need to maintain a mapping per CPU type and per crate type.

One of the main drawbacks of the PCI-based front-ends in our environment is the relative fragility of the connections when compared to real industrial solutions. Indeed, the PCI connection uses the Printed Circuit Board (PCB) tracks directly to make contact with the backplane; there is no connector soldered on it. The CompactPCI standard improves robustness as it specifies connectors for both the modules and the backplanes.

5.3.4 PXI

PCI eXtension for Instrumentation (PXI) is the equivalent to VXI for PCI. However, contrary to VXI, which was based on an already rugged industrial standard, PXI adds that aspect to PCI (see drawbacks in the PCI chapter). In addition, PXI defines lines for triggering and clock distribution.

The PXI support in BE-CO is quite marginal, as this platform is mainly used together with LabVIEW. A few systems have been deployed operationally for analogue signal acquisition in the LHC and kicker controls in other accelerators.

5.3.5 PCIe

With every evolution bringing higher performance, the parallel buses, such as PCI, fell out of fashion due to inherent limitations such as power consumption and bandwidth limitations (half-duplex shared bus and timing issues). Modern standards are based on serial buses and PCI Express (PCIe) is one of them. PCI Express supersedes PCI in many aspects including higher throughput, lower I/O pin count and native hot-plug feature. Similar to the other standards described above, the PCIe specifications are maintained by organisations and/or interest groups (PCI-Special Interest Group (SIG) for PCIe).

The key difference between PCI and PCIe is that PCIe is based on point-to-point links i.e. there are separate connections between the host and the different modules rather than having a bus shared among all modules. This topology brings many advantages such as concurrent access, no bus arbitrations, less EMC noise and lower power consumption. Furthermore, for high-bandwidth applications, one can combine several lanes to form a link (or interconnect). The standard defines slots and connectors with widths from 1 to 16 lanes (32 in the latest versions). So, for the version 1.x, which supports up to 2.5 Giga transfers per second (2.5 GT/s), it goes from 250 MB/s with one single lane to 4 GB/s if one uses the 16 lanes.

Physically, each lane is made of two differential pairs to provide the full-duplex byte stream. Of course, the PCIe connector contains more pins for power supplies and a few control signals. A 1-lane connector contains 18 pins but the 4-lane contains only 32 pins, which is easier to handle and more economical than the high pin count found in VME and PCI. It is important to know that a module with fewer lanes can be installed in a slot supporting more lanes. Indeed, the protocol negotiates the maximum number of lanes to be used between the host and the modules. This feature gives more flexibility when one selects a backplane configuration. As for the PCI bus, an explanation of the PCIe hardware protocol and its different layers is too complex and outside the scope of this document.

BE-CO has supported PCIe for a few years in the front-end layer, offering Kontron computers with several PCIe slots with widths up to 16x.

One advantage of a serial bus over a parallel one is that it is possible to put the bus on a cable or even a fibre. This opens up the possibility to physically distribute modules of a front-end computer outside of the computer enclosure. Today, this possibility is neither supported nor used by BE-CO.

5.3.6 MicroTCA4

Every 5-10 years, the technologies used in the front-end computers have to be re-assessed, both in terms of the evolution of our needs, as well as the market. Hardware module and platform obsolescence means that the group needs to perform regular technology and market surveys in order to ensure that we are still able to buy hardware. A study completed in 2018 indicated that microTCA.4 is an interesting platform that BE-CO aims to fully support by 2021.

Micro Telecommunications Computing Architecture (mTCA).4 is a subsidiary specification of mTCA, which itself is a standard that was originally intended to work with Advanced Telecommunications Computing Architecture (ATCA) systems, aka PICMG 3.x. For our applications, ATCA is not required and, mainly due its huge form factor, its price is prohibitive. The mTCA backplane is made of high-speed serial links in a star architecture, plus a few lines directly connecting the individual slots. The centre of the star is the MicroTCA Carrier Hub (MCH). The MCH provides the connectivity between the cards (aka Advanced Mezzanine Card (AMC)) and is similar to a switch. In addition, the MCH manages commodities such as the voltages, fans, diagnostics, etc. mTCA provides reliability by supporting redundancy and diagnostics. The power supplies are redundant and the AMCs are hot-pluggable with power-requirement management by the MCH; the AMC tells the MCH how much power it needs and, if it accepts, the MCH sends the power on the backplane. For the diagnostics, Intelligent Platform Management Interface (IPMI) is used and can be exposed, for example, with Simple Network Management Protocol (SNMP).

mTCA leaves many details open; for example, one can use Ethernet, PCIe, Serial RapidIO, etc. on the serial links. Such openness limits the interoperability and therefore extensions are defined. mTCA.4, the mTCA extension for physics, further constrains the platform with specific definitions for the backplane, the cards, clocks and trigger lines, etc. For the backplane, out of the 8 lines available between each AMC slot and the MCH, mTCA4 imposes lines 0 and 1 to be Gigabit Ethernet, lines 2 and 3 are reserved for the daisy chain and the 4 remaining lines are dedicated to the FatPipe, which is PCIe x4. mTCA.4 also adds the possibility to use Rear-Transition Modules (RTM) which can be useful when the board real-estate (148.8 mm by 181.5 mm) is too limited and/or when more front-panel connectors are needed.

5.3.7 Future Technologies such as PXIe

As the reader might have guessed, PXI Express (PXIe) is the PCI Express equivalent to PXI. In its latest edition, the PXIe standard specifies system bandwidth to up to 24 GByte/s.

PXIe is seen as a potential future platform that the BE-CO group will support. At the time of writing the 2019 edition, there is a collaboration project between BE-CO and EN-SMM in order to provide a prototype of a suitable PXIe offer, i.e. a crate with power supply and a system board running the BE-CO software stack, by 2021.

5.4 CO-Supported Electronic Modules aka CO Kit

Most of the controls use cases require a hardware component in their implementation. Indeed, particle accelerators are analogue and we need to supply and monitor analogue signals to and from the accelerator equipment. However, the Control System is digital and the hardware modules provide the interface between the two worlds.

As many use cases can be decomposed into common building blocks, the BE-CO group provides a catalogue of ready-to-use, supported hardware modules. For each of them, we offer the electronic board, its kernel device driver and access library, and a test program. For some of them, we optionally provide the FESA class (see section 11.2). Thanks to this approach, the Control System users save integration time and the procurement and support is centralised.

The BE-CO hardware kit provides generic functions such as:

- Time-to-Digital Converter (TDC);
- Analogue-to-Digital Converter (ADC);
- Analogue function generators;
- Pulse production and pulse delay;
- Analogue and digital Input/Output (I/O);
- Fieldbus controllers for various bus standards;
- Etc.

For most of the functions, we have a range of modules with different characteristics e.g. we have about 10 different ADC boards with different sampling speeds and resolutions.

Some modules with more CERN-specific functions are developed in-house, but, whenever possible, we buy COTS from commercial vendors. If a piece of hardware is specific to a particular partner group, it makes sense that they design, integrate, and support the module themselves. Of course, we recommend that whenever common needs are identified, CO-standard, supported modules are used.

In recent years, we have pushed the reusability and modular approach even further by factoring out the common features found in almost all of the modules into a carrier board. The carrier board provides the bus access (VME, PCIe, or PXIe), the memory, and an FPGA to implement the board's logic. Mezzanine boards are then used to fulfil the specific (mainly analogue) part.

In order to split the labour in a rational way, BE-CO provides the infrastructure i.e. carrier boards, whilst the partner groups may design the mezzanines. In general, carrier boards take longer to design and require a different set of skills than those needed for the mezzanines and this reinforces the decision to distribute the work.

As we want to interact as much as possible with industry, we needed to choose a standard between the carrier board and the mezzanines. Vendors want to reduce economic risks and a standard helps in that respect. When the choice was made (in around 2010), the only one available was an ANSI/VITA standard called FPGA Mezzanine Card (FMC) [2, 14].

5.4.1 FMC Modules

FMC is an agnostic standard that brings a modular I/O approach to FPGA design. The BE-CO group has decided to use the Low Pin Count (LPC) connector, which has 160 pins, for cost reasons. Out of the 160 pins, 68 are user-defined. Alternatively, the user-defined pins can be used as 34 differential pairs. The High Pin Count (HPC) version of the connector offers more lines and, in addition, a few serial transceiver pairs and clocks. It is worth noting that the HPC and LPC connectors are mechanically compatible. Figure 5.3 depicts an FMC mezzanine where one can see the FMC connector on the right and the application-specific external connectors on the left.

Compared to other carrier/mezzanine options such as V-MOD (also used in the BE-CO hardware kit), the FMC carrier is more sophisticated. As already mentioned, the carrier hosts an FPGA that contains the digital logic for the specific function of the board. Note that the carrier/mezzanine pair cannot be exchanged without altering the FPGA code and risking burning outputs in the FPGA and the mezzanine's components. To avoid this situation and provide mezzanine identification, the standard reserves pins for an Inter-Integrated Circuit (I2C) bus.

The CERN-designed carrier boards feature an FPGA (Xilinx Spartan-6), 256 MB of RAM (2 x 256 MB for the VME version), and a Small Form-factor Pluggable (SFP) socket that can be used, for example, to plug a 1 Gb/s optical transceiver into the board. The VHDL code to access the memory and the external connectivity (bus, transceiver) is readily available so that only specific business logic has to be implemented. One of the target applications of the optical transceiver at CERN is the integration with the White Rabbit

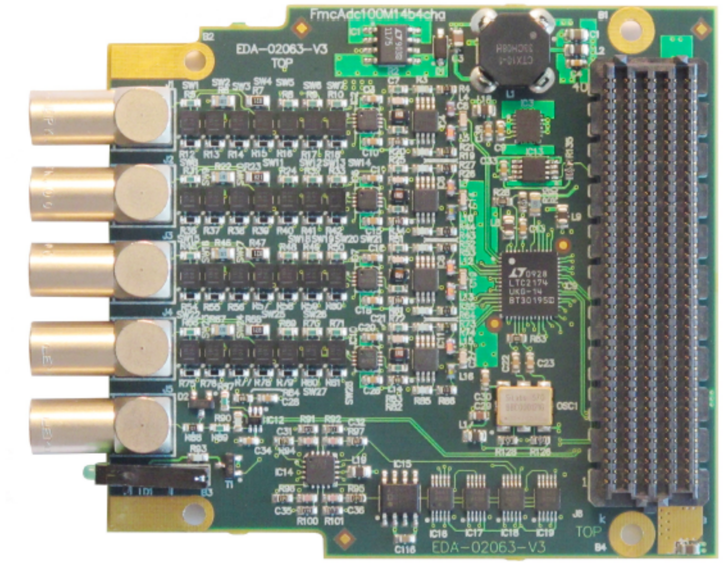


Figure 5.3: Analogue-to-Digital Converter (ADC) based on the FMC mezzanine format

network, but other applications are possible. The FPGA configuration Bitstream can be uploaded from the bus interface, allowing remote re-configuration of the carrier board.

BE-CO supports carrier boards with bus interfaces to VME and PCIe called Simple VME FMC Carrier (SVEC) and Simple PCIe FMC Carrier (SPEC) respectively. In addition, the EN-SMM group designed and supports a PXIe carrier (Simple PXIe FMC Carrier (SPEXI)). The VME version gives access to the VME crate's RTM and provides for two mezzanine sites, whilst the other carriers have only one. In addition to the carrier boards, BE-CO designs mezzanines for generic functions such as ADC, TDC, fine delay and digital I/O.

We observed the advantages of the carrier/mezzanine approach and specifically FMC, during the redesign of the WorldFIP master board. Instead of designing a complex PCB, we only had to develop the FPGA code to implement the communication stack and a mezzanine to convert the signals between the FPGA digital levels and the WorldFIP standard.

On the other hand, the overall cost of the FMC approach is higher if modules have to be produced in large quantities. The real estate on the mezzanine board is also fairly limited with a standardised size of 69 mm by 76.5 mm; a problem that can be solved in VME crates by using a Rear-Transition Module (RTM) even though not all of the pins are available.

5.4.2 Non-FMC Modules

Even though the advantages of the FMC approach are clear, there are two situations where we still rely on single board implementations: for the COTS modules and older CERN-made boards. For the latter, we still support several boards that were designed before the FMC era. For example, the Current Timing Receiver family (CTRx) is based on single PCB boards available for different backplanes (VME, PCI). In addition, we still support modules that were re-designed around the year 2000, such as the function generators

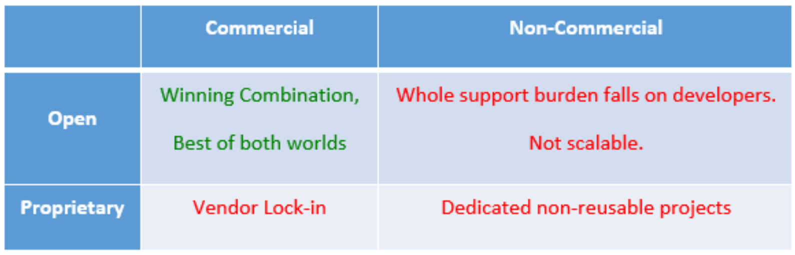
(CVORB, CVORG) and the PCI version of the MIL-1553 bus controller (CBMIA). In 2019, in the BE-CO group, only boards which are designed to be drop-in replacements for older boards are foreseen to be developed outside the FMC approach.

There are also situations where a CERN-driven design is not appropriate. It can be because there are already perfect solutions available as COTS or because the requirements are such that they would require a level of expertise that we do not have in the group. For this reason, we also support COTS modules. Most of the COTS modules are Analogue-to-Digital converters, as this function is very common and the market has plenty to offer, but as one reaches the highest sampling rates and resolutions, these modules require experts in the domain to develop. To cite a few, our catalogue contains the Agilent/KeySight digitisers with sample frequencies up to 8 GSa/s and the more modest INCAA VD80 at 200 kSa/s.

5.4.3 Development Philosophy

For many years, our modules were procured in two different ways. They were either designed, manufactured, and tested at CERN or commercial boards, based on proprietary designs. Neither solution was fully satisfactory. The amount of work required to produce and support our own boards quickly exceeded our human resources. With COTS, we found ourselves in vendor-locked situations, being unable to buy discontinued products and exposed to price hikes. It was also difficult to influence priorities when it came to new features and bug fixes.

We can define two orthogonal aspects for our development policy. One axis is the open-source versus proprietary design and the second is whether we rely on companies to provide part of the service (commercial versus non-commercial). Table 5.1 gives the advantages and disadvantages of the four possible combinations.



	Commercial	Non-Commercial
Open	Winning Combination, Best of both worlds	Whole support burden falls on developers. Not scalable.
Proprietary	Vendor Lock-in	Dedicated non-reusable projects

Table 5.1: Pros and cons of different development philosophies

It is clear that an open-source commercial approach is preferable. In this way, we can sub-contract part of the design, manufacturing, testing and support to companies for a fee, whilst staying in full control of our products and avoiding vendor-locking.

In order to counteract companies' risk aversion, we developed the CERN Open Hardware license,[5] a legal framework that allows people to share electronic designs [8, 14].

[5] https://www.ohwr.org/licenses/cern-ohl/license_versions/v1.2

5.5 Fieldbuses

A fieldbus is an industrial real-time network used to control distributed agents that are typically of limited power. Compared to Local Area Networks (LANs), fieldbuses are used at lower level to connect intelligent actuators and sensors to more sophisticated supervision systems such as a Front-End Computer. Figure 1.6 depicts the typical layout of the lower tier based on a fieldbus. The BE-CO Control System has been based on fieldbuses since the very beginning and, therefore, it is not surprising that we support several of them.

5.5.1 MIL-1553

Based on a military standard (MIL-STD-1553-B), MIL-1553 has been used in the Control System since the middle of the 80s. It was initially chosen for its robustness; it was also used in airplanes and ships. CERN's implementations of MIL-1553 have diverged from the standard over time, as it was adapted to our specific needs. In hindsight, with such complex maintenance of critical installations in the injector complex, it would have been better to have remained with the standard.

MIL-1553 is a simple multi-drop bus based on a 5-volt Manchester-encoded differential signal, sent over a shielded twisted pair. A bus has one master, known as a Bus Controller (BC), and up to 32 slaves, connected through Remote Terminal Interfaces (RTIs), that answer to the master in half-duplex with a throughput of up to 1 Mbit/s. To ensure galvanic isolation and integrity of the bus, the agents are AC-coupled to the bus using transformers. The bus is simply terminated by a 50-Ohm resistance in a stopper. MIL-1553's messages are 20 bits long and carry a 16-bit payload. The message integrity is ensured by a 3-bit synchronisation field and a simple parity bit. There is no specific hardware for diagnostics, meaning that we have to rely on software traces provided by the driver. According to the standard, the bus is deterministic but, as we have many different slaves all with slightly different firmware, the MIL1553 installations have become tricky to diagnose. Furthermore, due to the lack of synchronisation service from the bus, typical installations have a MIL1553 bus for data control and acquisition, as well as a set of copper cables to transport between 1 and 5 timing pulses for the action synchronisation (e.g. start capacitive charge, read current value, etc.). The misconfiguration of timing to either trigger bus transactions or trigger equipment's actions leads to complex synchronisation issues. This sub-optimal setup has a lot of room for improvement and better solutions are described in the next two sub-chapters.

MIL-1553 is now a deprecated fieldbus, but it is still used to control critical elements in the injector complex. In 2019, some bus controllers remain installed in operational front-ends and therefore we have to support MIL-1553 until these installations are renovated. For this reason, and to regain full-control over the black box that the VME module had become, we undertook the design of PCI master board in 2011; the so-called CBMIA.

5.5.2 WorldFIP

With the construction of the LHC, it made sense for CERN to standardise the fieldbuses that were to be used in the new accelerator. In 1995, a working group studied the market for available radiation-tolerant fieldbus solutions. In 1996, the group gave the recommendation to use the WorldFIP fieldbus for applications in the LHC tunnel. Initially a French effort from the 80s, World Factory Instrumentation Protocol (WorldFIP) is a European open standard (EN50170). Despite the fact that the WorldFIP design was not made with radiation-tolerance in mind, tests showed a good performance of the node under radiation, due to its relative simplicity and the chip technology used in the early versions (0.6 mm), and this is why it was finally proposed as the radiation-tolerant fieldbus for the LHC. Retrospectively, the choice was correct as the WorldFIP installations (>10'000 nodes installed all around the LHC tunnel) have shown good robustness and reliability.

WorldFIP is a real-time fieldbus with a linear multi-drop topology. It is based on a Master/Slave architecture where the bus's access right is centralised by the Master, which continuously distributes the access token to the slaves in a cyclic manner. The Master runs on a dedicated processor that does not depend on the Front-End host computer; this ensures the real-time performance. The Master can also receive an external timing signal (from GMT) to which it synchronises the beginning of each cycle; like this WorldFIP can provide distributed synchronised communication. Figure 5.4 depicts a typical CERN topology. Although not used at CERN, WorldFIP supports multi-master configurations. The protocol foresees four bit rates: 31.25 kb/s, 1 Mb/s, 2.5 MB/s, and 5 Mb/s. The 5Mbps option is only available as a prototype and is not currently used by any equipment groups. The WorldFIP standard also specifies the maximum cable lengths and number of slaves per bit rate e.g. at 1 Mb/s, the maximum bus length is 1 km of copper with a maximum of 255 nodes.

Repeaters are used to either convert back and forth from copper to optical fibre or to extend the maximum length of the bus. In the LHC, we typically use the optical fibres to descend into the shaft where only straight-line paths are needed and where there are no agents. In the tunnel, we use shielded twisted pair copper cables. The signal on the pair is a 5-Volt differential with Manchester encoding and the agents are AC-coupled through transformers ensuring the galvanic isolation from the rest of the bus.

WorldFIP uses a fixed-length cycle called a macrocycle. At CERN, depending on the application, the macrocycle is between 20 milliseconds and 1 second long with a maximum frame size of 124 data bytes. The macrocycle is divided into the deterministic traffic, the event-type traffic, and the network management services. The structure of the deterministic traffic is established when the system is put in place and is continuously repeated without any possible modification. During this phase, the master sends a question frame to the agents indicating which agent it is talking to, and whether the agent should produce an answer or consume incoming data. When sending the answer, an agent can raise a bit to indicate that more data are to be sent and that a slot in the event-type traffic should be reserved for that transfer. The master can then allocate a slot for that message exchange in one of the upcoming event-type phases, but not necessarily the next one. Figure 5.5 illustrates a typical WorldFIP macrocycle and that the event-type traffic and the network management services are optional and can be omitted according to the application's needs. At CERN, some installations do not have event-type traffic but, as diagnostics of such installations is of paramount importance, all of the WorldFIP installations have their bus terminated by a special agent called the Diagnostic Agent for WorldFIP (FIPdiag). This diagnostic agent is programmed to simply send back the data it received, but it is connected to the bus through an attenuator. During the network management phase, in addition to polling the presence of all agents, the FIPdiag is accessed and if its answer matches the

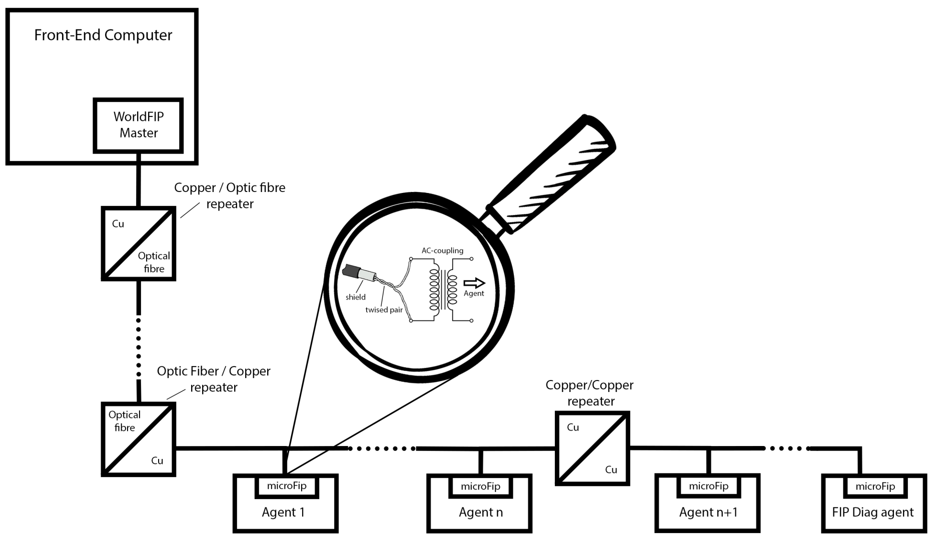


Figure 5.4: Typical CERN WorldFIP installation

question, we can consider that the bus installation is operational as the other agents are closer to the master and connected without additional attenuation.

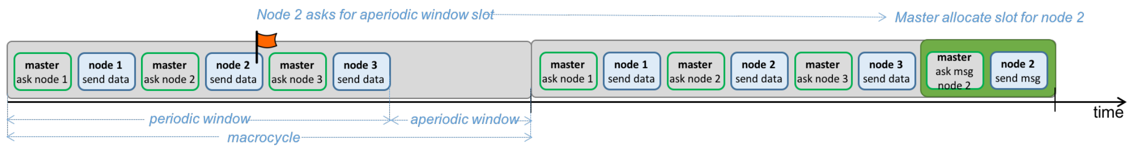


Figure 5.5: WorldFIP macrocycle

In recent years, it was decided to in-source both the WorldFIP master and the slave chip (the so-called microFIP). The main reason to design a new slave chip, the nanoFIP[6], was that the latest versions of the microFIP were based on newer chip technologies (0.5 mm) and the accidental radiation tolerance had been lost. Similarly, the master board and accompanying software libraries could not follow the technology evolution such as the arrival of multi-core CPUs in the FECs. Combined with the decreasing support from the industry and the investment CERN had made in the technology, a project[7] to design and produce a new WorldFIP master board was launched [45]. The new master is based on the FMC approach and the SPEC PCIe carrier board described in section 5.4.1. The FPGA code uses the MockTurtle FPGA framework (see section 9.2 for details) that allows

[6]`http://www.ohwr.org/projects/nanofip/wiki`
[7]`http://www.ohwr.org/projects/masterfip/wiki`

the high-level part of the protocol to be implemented in C and run on soft-cores while the low-level part (e.g. WorldFIP serializers/deserializers and Cyclic Redundancy Check (CRC) computation) is kept in VHDL. Even though the new Master is a stripped-down version of the WorldFIP protocol (e.g. no master redundancy), it is fully compatible with all CERN applications. The new module has been massively deployed during the LS2 period.

After 10 years of operation, the WorldFIP installations have proven to be robust and reliable and we plan to support the technology until the end of High-Luminosity Large Hadron Collider (HL-LHC).

5.5.3 Powerlink

Support, in terms of fieldbuses, needs to evolve to cover the future needs of the equipment groups. For new applications targeting the HL-LHC era, we need to introduce and centrally support more performant and interoperable solutions, since the current bandwidth is a bottleneck, being limited to 2.5Mbps.

The replacement of WorldFIP, which did not achieve the expected success in industry, will most probably be Ethernet-based. Currently dominating the market are Industrial Ethernet fieldbuses at 100Mbps, and several options are available such as EtherCAT, EthernetIP, POWERLINK, and Profinet. The latter, from Siemens, is already used at CERN in zones without radiation constraints, for example in cryogenic and collimator controls. For the areas with high-radiation levels, none of these fieldbuses currently have off-the-shelf radiation-tolerant solutions and a CERN-made design, applying radiation-hardening techniques, is still required. It must be taken into account that the additional complexity of Ethernet, with its many layers, may make it difficult to achieve the required radiation tolerance.

In the space domain, organisations, such as the European Space Agency (ESA), require a radiation-tolerant high-bandwidth fieldbus and rely on SpaceWire. This option was also investigated but SpaceWire lacks interoperability with off-the-shelf equipment and, being designed for spacecrafts, imposes constraints on the distance between nodes and total segment length that are incompatible with large particle accelerators such as the LHC.

For these reasons, solutions such as POWERLINK, a very light protocol built on top of Ethernet's physical layer, are more appropriate. Furthermore, POWERLINK software and firmware is both free and open source. Nevertheless, the master-based half-duplex communication (aka "you speak when spoken to") can be seen as a limitation.

In addition, for cases requiring sub-nanosecond synchronisation outside radioactive areas, the group offers White Rabbit, as explained in section 5.6. However, White Rabbit is not technically a fieldbus, as it does not define the upper layers of the OSI model. For example, it does not define cyclic exchange of data between several nodes; one would need to add a protocol on top of White Rabbit for those tasks, such as White Rabbit Trigger Distribution (WRTD), as described in section 5.6.1. Furthermore, due to White Rabbit's very high complexity, any radiation-tolerant design would be very challenging to make.

POWERLINK is very similar to WorldFIP. It provides macrocycles in the order of milliseconds and synchronisation in the microsecond range. Its protocol is simple; the master broadcasts a frame to the slaves, the concerned slave identifies with the request and replies on the bus. This half-duplex mode ensures that there are no collisions and no loss of determinism, due to the lack of contention.

Thanks to the use of standard 100Mb/s Ethernet, the non-radiation-tolerant hardware, such as the Network Interface Controllers (NICs), can be bought off-the-shelf. Cables are standard copper Ethernet cables and switches are agnostic to POWERLINK. All of the diagnostic tools to monitor and analyse the POWERLINK traffic are also standard. For longer distances, for example from the surface to the LHC tunnel, optical fibres must be used, but again, standard optical-to-copper converters are available.

We plan to develop two slave FMC mezzanines, one for the radiation-free areas and a second radiation-tolerant version. As for WorldFIP, the target radiation tolerance for the electronics is 400 Gray. For the implementation of the physical layer, it should be possible to use COTS chips, with the appropriate radiation tolerance. The POWERLINK protocol will be implemented in a radiation-tolerant FPGA. Figure 5.6 depicts the overall architecture of the FMC. As for WorldFIP, the components and the final design will be validated by radiation test campaigns. The envisaged topology requires the slaves to be daisy-chained. Therefore, each slave needs to act as a dual-port switch. While this additional requirement is not complex to implement, one challenge is to ensure that forwarding is still active, even when the slave is broken down or not powered.

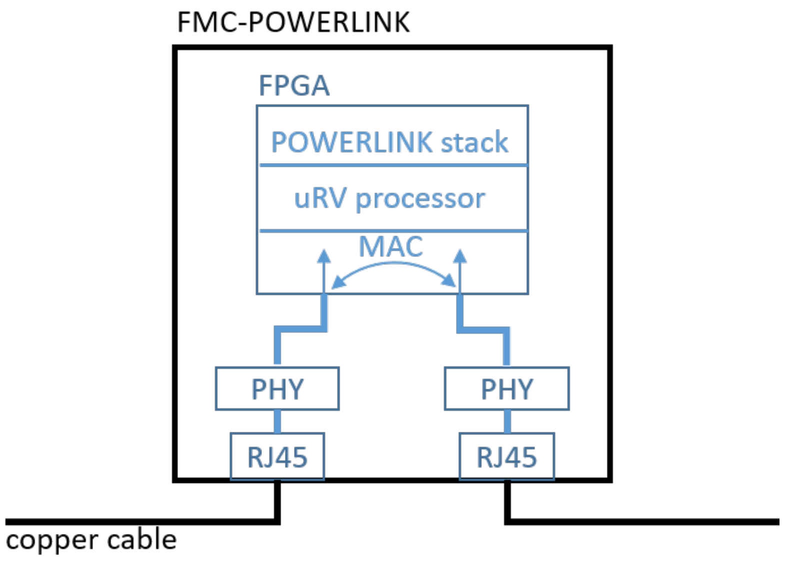


Figure 5.6: POWERLINK FMC architecture

On the master-side, which always stays in radiation-free areas, two possibilities exist. PLCs with native support for POWERLINK can be purchased from B&R but, as the group relies on FECs to control fieldbuses, a PCIe master will also be developed. Even though the master is a standard network interface card, it needs to be able to receive a synchronisation pulse from the timing network in order to align macrocycles across networks, as implemented for CERN's WorldFIP networks.

5.6 White Rabbit

Started in 2009, White Rabbit is a fully deterministic, Ethernet-based network solution to design and implement distributed, hard real-time systems [59]. The main goals of the project were to provide a technology which solves recurrent problems and limitations in the current controls infrastructure. Firstly, we wanted the new generation of the GMT network to have a much higher bandwidth to avoid multiple networks and improved diagnostics thanks to a full-duplex network. In addition, we wanted to provide a solution for the synchronisation problems we have with MIL-1553 installations, for which both data and synchronisation links are needed with precise cross-configuration.

White Rabbit adds two extra services on top of Ethernet. Firstly, it provides a common notion of time across all nodes with nanosecond precision. Secondly, it offers guaranteed upper bound latency in message delivery. Thanks to these two features, it is possible to design distributed systems that will produce synchronised distributed actions, provided the command is given early enough.

The common notion of time is provided thanks to an improved version of IEEE 1588 (aka Precise-Time-Protocol (PTP)). The White Rabbit protocol evaluates the transmission time so that the slave nodes can compensate the time needed for a message to reach them. Figure 5.7 shows the PTP messages exchanged to synchronise the slave time with the master time. At *t1*, the master sends a SYNC message that, upon reception, triggers the recording of the current time *(t2)* in the slave. The master then sends a `FOLLOW_UP` message containing the value of *t1*. *t2 - t1* is the sum of the transmission delay and the offset between the master clock and the slave clock. In order to isolate the clock offset, the slave sends a `DELAY_REQ` at *t3* to the master. The master precisely records the arrival time of the message *(t4)* and sends the value in a `DELAY_RESP` message. The slave now has all the variables required to determine its offset with respect to the master clock, which is *(t2 - t1 - t4 + t3)/2*. While PTP has a precision of 1 microsecond, the White Rabbit version offers a 1-nanosecond precision. Concretely, this means that no matter how long the fibre is between the nodes, they will have exactly the same Coordinated Universal Time (UTC), ticking in phase. This improvement of IEEE 1588 is itself in the process of being integrated into revision 3 of the standard, under the high-accuracy profile.

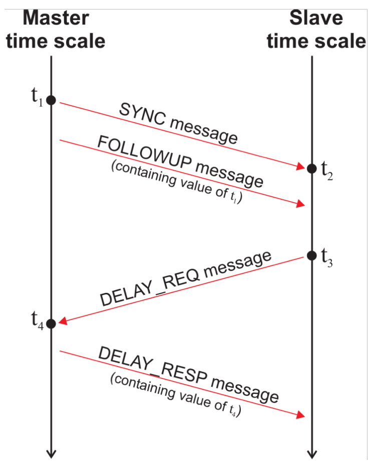


Figure 5.7: PTP message exchange

The message delivery's upper-bound latency can be guaranteed thanks to a custom-designed White Rabbit switch for which we can compute the delay it introduces. Furthermore, the White Rabbit switches support 802.1Q, an optional part of Ethernet, which allows messages to have priorities via a Priority Code Point. Figure 5.8 shows how the 802.1Q header is inserted into an Ethernet frame. The figure also illustrates the Priority Code Point (PCP) field which carries the message's priority. For a given White Rabbit network and for the highest-priority messages, knowing the longest path and the number of switches allows us to determine the upper-bound latency for any such message in the network. In 2019, the largest White Rabbit network deployed at CERN has an upper-bound latency of 100 microseconds. By combining the two features of a common notion of time and upper-bound latency, we are able to produce pulses in phase, with a 1-nanosecond precision, that are distributed around the LHC.

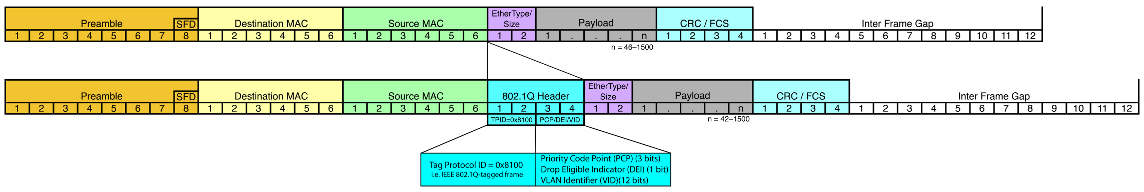


Figure 5.8: 802.IQ header in an Ethernet frame[8]

White Rabbit can also be used to do more than producing pulses. For example, we can use it to distribute clocks in different points of an accelerator using the distributed Direct Digital Synthesizer (DDS) approach, or we can also use it to distribute synchronous data such an accelerator's B field values. Also, since White Rabbit is based on Ethernet, it also inherits its advantages and its weaknesses. For example, we cannot reach extremely low latencies. On the other hand, we benefit from many powerful diagnostic tools and it is straightforward to obtain node diagnostics using the SNMP protocol.

White Rabbit is foreseen to be used at CERN for several projects such as the distribution of the main-magnets' B-field and the new OASIS triggers' distribution. In the long term, it is foreseen that White Rabbit will gradually replace the ageing General Machine Timing (GMT) distribution.

5.6.1 Trigger Distribution

Producing pulses in phase, with a 1-nanosecond precision, is exactly what the LHC Instability Study Triggers (LIST) does. This application reproduces pulses generated in one point in the LHC in several other points at CERN [63]. Whenever an event worth studying occurs, a pulse is sent to one of the LIST input modules (a specialised version of the FMC TDC). The incoming pulse is then time-tagged and the tag is sent over the White Rabbit network. Thanks to the features explained above, we can then reproduce the trigger pulse in many other locations with a minimum delay of 100 microseconds, which is the upper bound latency of the network (the maximum delay is user configurable). Figure 5.9 depicts the network used to distribute the triggers. In 2019, the LHC Instability Study Triggers (LIST) network has 7 operational nodes but can be extended with additional nodes if necessary.

[8]based on work by Bill Stafford (own work) [CC BY-SA 3.0], via Wikimedia Commons.

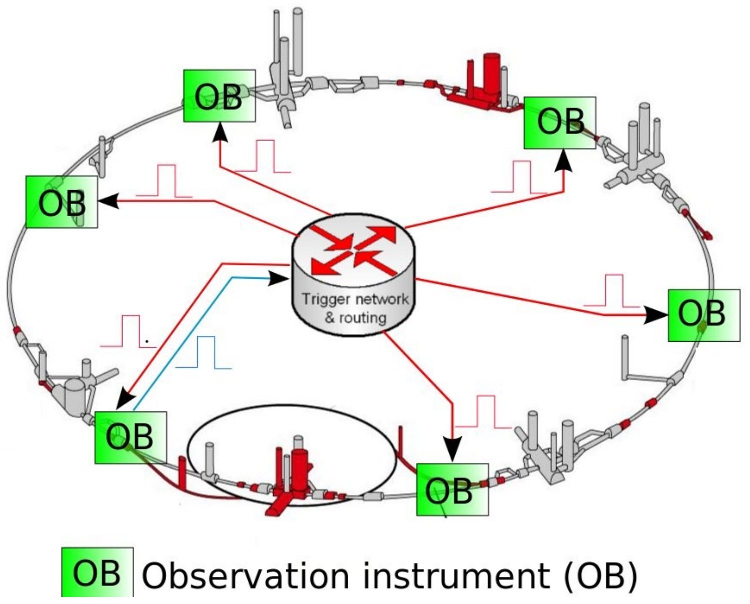


Figure 5.9: White Rabbit network for LHC Instability Study Trigger distribution

Building on the experience gained with LIST, the WRTD project was launched, aiming to generalise the concepts required to build a network of distributed instruments. The LIST API was very specific to the use-case and the implementation assumed that the underlying hardware was a SVEC with a TDC and a fine-delay. Ultimately, WRTD should become an extension of the Interchangeable Virtual Instrumentation (IVI) standard, which industry can apply in their products, such as digitizers. IVI is an industrial standard managed by a eponymous foundation, defining a classification of instruments, as well as a large API to control them. Manufacturers package IVI drivers implementing the API with their instruments. It should be noted that IVI does not make any assumptions about the underlying layers, and as such, WRTD does not require the transport layer to be White Rabbit. In theory, any transport could be used, on the understanding that other types of network do not offer a common notion of time and upper-bound latency.

WRTD can be used in two different ways. In the first case, the timestamp of an event is sent across the network and the receiving nodes roll back the acquisition history to isolate the correct data. In the second case, the emitting node sends a timestamp in the future, taking into account the network latency in order to produce synchronised pulses at the receivers. The latter case will be applied in the OASIS triggering system, as explained in chapter 23.

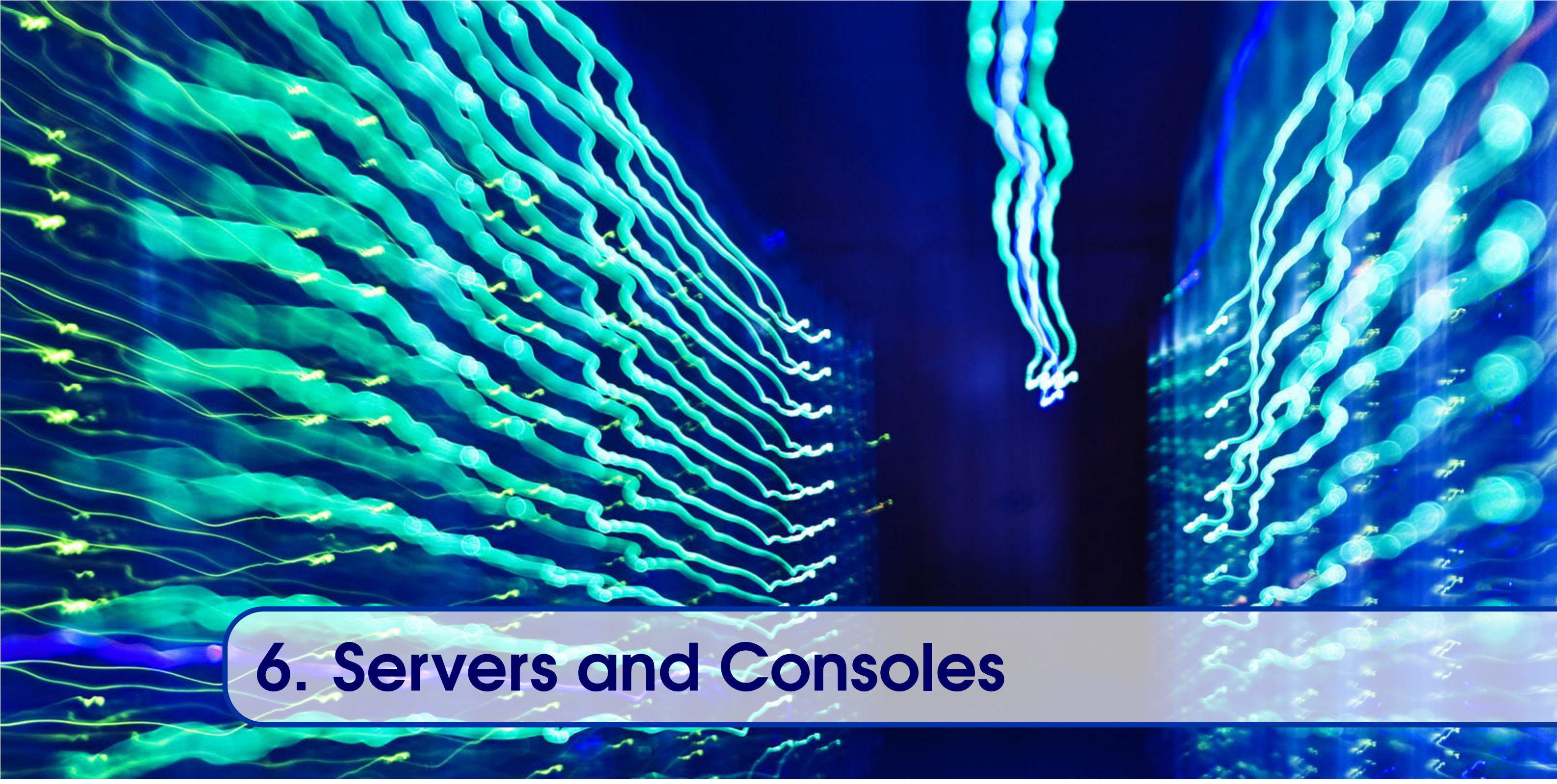
WRTD is a technology designed to support large networks and therefore appropriate monitoring and diagnostics are available. The number of occurrences of an input or output are logged, as well as the number of missed messages, typically because they arrive too late. In addition, statistics such as average frequency of messages, minimum, average and maximum latency of the various network paths are calculated.

The WRTD message format is based on an LAN eXtensions for Instrumentation (LXI) standard, as depicted in figure 5.10. In 2019, only one event per message is sent in the header. In the future batching would be possible as the message payload is available, however, this is not currently compatible with the LXI standard.

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0x4c ('L')								0x58 ('X')								0x49 ('I')								0x00							
4	32	Event ID [0:3]																															
8	64	Event ID [4:7]																															
12	96	Event ID [8:11]																															
16	128	Event ID [12:15]																															
20	160	Sequence number																															
24	192	Event timestamp seconds (32 lower bits)																															
28	224	Event timestamp nanoseconds																															
32	256	Event timestamp fractional nanoseconds																Event timestamp seconds (16 upper bits)															
36	288	0x00								0x00								0x00								0x00							

Figure 5.10: WRTD event message

In 2019, the focus is on producing a tabletop demonstrator of two oscilloscopes triggered by same source but with different cable lengths (50m + 2.5km), in order to validate the new reference designs and APIs and optimise the latency of the White Rabbit network. In the near future, a SVEC-based VME module with two FMC-ADC mezzanines (100MSa/s) a well as two VME modules with either two TDCs or two fine-delays will be developed. Finally, there are also plans to develop another, faster, ADC with a sampling frequency of up to 1GHz.



6. Servers and Consoles

6.1 Server Platforms

The server infrastructure provides three main functions: Network File System (NFS) file servers, front-end computer boot servers and high-level application servers for Injector Control Architecture (InCA), CERN Experimental Area SoftwAre Renovation (CESAR), WinCC OA, etc. In the first two cases, the amount of storage is a priority while, for the application servers, memory and CPU power are more important. In 2019, we have four different solutions, the older generations, based on HP servers, which should be phased out before the end of their 5-year warranty period, and the latest technology, for which we insisted on a more generic solution, independent of a specific manufacturer.

The key characteristic of the server infrastructure is the necessity for 24/7 availability. In addition, and contrary to other aspects of the Control System, the main maintenance window is very small, being strictly limited to 3 days at the beginning of each year. At all other times, interventions have to be as fast and transparent as possible and are applied only to non-critical infrastructure services. The system must report issues so that interventions can be scheduled.

For the storage-oriented solution, we used to use the HP ProLiant DL380 family but we are gradually migrating to quad servers combined with Just a Bunch Of Disks (JBOD) enclosures. Similarly for the application servers, the HP ProLiant BL460 family is progressively being replaced by servers in the quad form-factor.

All solutions are rack mountable, which allows scalability and ease of maintenance. Also, most of the components are duplicated and hot-pluggable, meaning that the Hard Disk Drives (HDDs), Solid-State Disks (SSDs), fans and power supplies can be exchanged without switching off the system. An advantage of the HP hardware is that CPUs, memory and storage are completely interchangeable between the two families, making spare-part management easier. On the other hand, the new approach, treating hardware as commodities, avoids a vendor lock-in situation.

The memory modules are based on Error-Correcting Code (ECC) to detect and correct errors. In addition, the memory has more capacity than advertised so data can be automatically relocated in case of failure.

6.1.1 HP ProLiant DL380

The HP ProLiant DL380 family is based on 2U enclosures. In the latest generation (GEN9), it supports up to 24 x 2.5-inch HDDs without extensions. If additional HDD space is required, PCIe connectivity to one or several external Redundant Array of Inexpensive Disks (RAID) Host Bus Adapters (HBAs) is available. The hard-disk drives (HDD) are server-grade disks spinning at 10’000 RPM with two sets of heads per disk to further improve the performance. For data protection, we mirror the HDDs using a RAID 1+0 configuration. In special cases, RAID 5 and 6 may also be applied to larger storage arrays. The DL380 family has 4 built-in LAN interfaces with a link speed of 1 Gbit, which can be increased by adding an optional 10 Gbit interface.

Two generations of the DL380 family are currently deployed in production; Generation 8 (GEN8) and Generation 9 (GEN9). In 2019, ten DL380 computers are still in use for Java application servers.

6.1.2 HP Proliant BL460

For the application servers, a high-density solution is required due to the high number of CPUs needed and the physical space constraints. Therefore, we decided to use the HP ProLiant BL460 solution, which is based on a 10U managed enclosure that can host up to 16 blades. In addition to the space efficiency, this solution is also more cost effective as one enclosure and 16 blades cost significantly less than 16 individual DL380 computers.

Each blade is a complete computer with CPU, memory and storage. The BL460 is CPU-power oriented with only two slots for disks and limited local storage (1.8 TB in RAID 1). The HDDs are the same as those used in the DL380 family. The blade enclosure provides power-supply redundancy; out of six available power supplies, up to three can be lost without degrading the service. The BL460 family has 2 built-in LAN interfaces with a speed of 1Gbit for generations G5 to G7 and 10 Gbit for generations GEN8 and GEN9. If needed, a blade can be replaced by an extension module where a PCI or PCIe board can be installed. The extension board is only visible to the adjacent blade and not all blades in the enclosure. This feature is used in one of our installations to distribute accelerator timing information locally.

In 2019, about one hundred blades are in operation and, as for the DL380 family, we have both GEN8 and GEN9. They are named after the system they are used for, such as cs-ccr-inca2, cs-ccr-cmw4... Figure 1.3 depicts an HP enclosure with 14 BL460 of different generations. Figure 6.1 shows the internals of a blade server.

6.1.3 Quads and JBODs

In 2017, we took the strategic decision to treat server hardware as a commodity. The new generation of server hardware is based on a 2U rack-mountable enclosure that can host

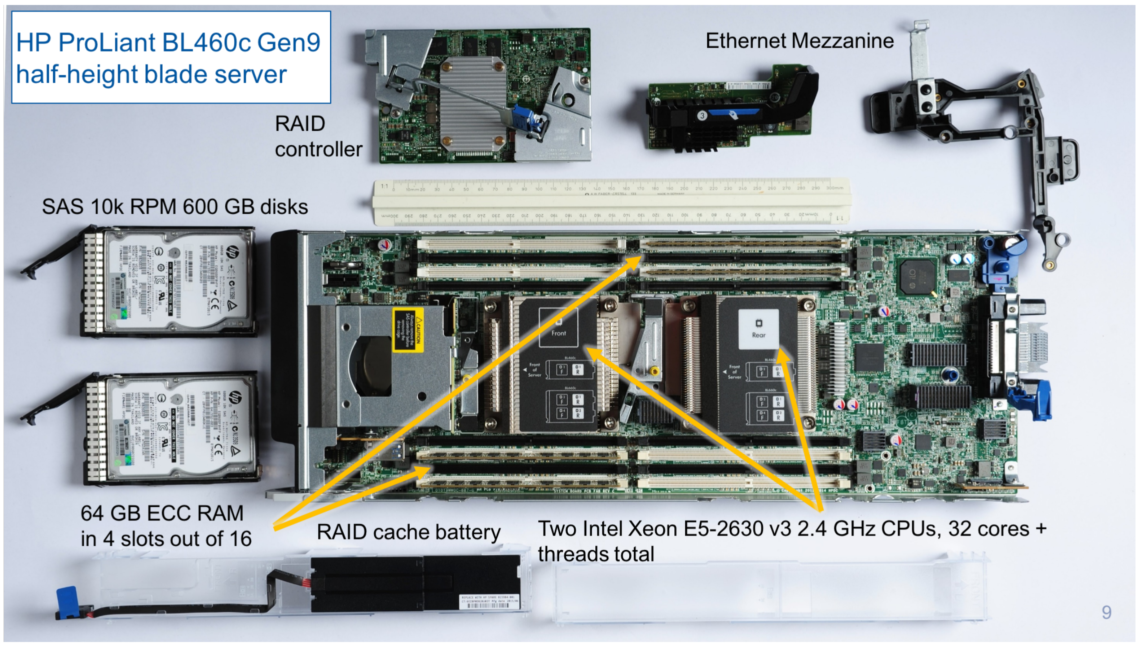


Figure 6.1: HP ProLiant BL460 - internal view

up to 4 servers, as shown in figure 6.2. We commonly refer to this hardware as a "quad server". Whenever a high storage capacity is required, we take the "Just a Bunch Of Disks" (JBOD) approach, connected to a quad server. The hardware is replaced as part of a 5-year lifecycle, with regular calls for tender and yearly purchasing.

The enclosure provides fewer services than its HP counterpart, in this case, only redundant power, ventilation, and mechanical support. Everything else is handled by the servers themselves, or external network equipment. A server is a double-CPU computer with 2 Ethernet connections. The first connection is dedicated to the system, while the second is used for IPMI monitoring and management. For the system connection, the highest bandwidth possible is required, and therefore, in 2019, the servers are connected to 10Gb/s Ethernet switches. On the other hand, the management connection doesn't have any speed requirements, but since it is used to administer critical systems, its connectivity is restricted to a few selected machines.

Since the new enclosures offer fewer services than the HP blade enclosures, which had integrated switches, the Information Technology (IT) department has now installed new 10Gb/s switches in the networking racks. This layout brings more complexity in terms of cabling, because each server needs its own connection, but is more straightforward. However, for the management connections, a switch is installed directly in the quad rack.

Every server has a few internal slots for drives, which are typically reserved for system installation and application software. In 2019, a server comes with 2 SSDs of 960GB that are configured in RAID 1. In some cases, additional SSDs are installed when only a moderate increase in storage is required. In the other cases, a JBOD enclosure is installed.

The JBOD enclosure is 4U and rack-mountable. They are purchased fully-populated with 24 3.5" SATA hard drives of 6TB each; giving the whole enclosure a raw capacity of

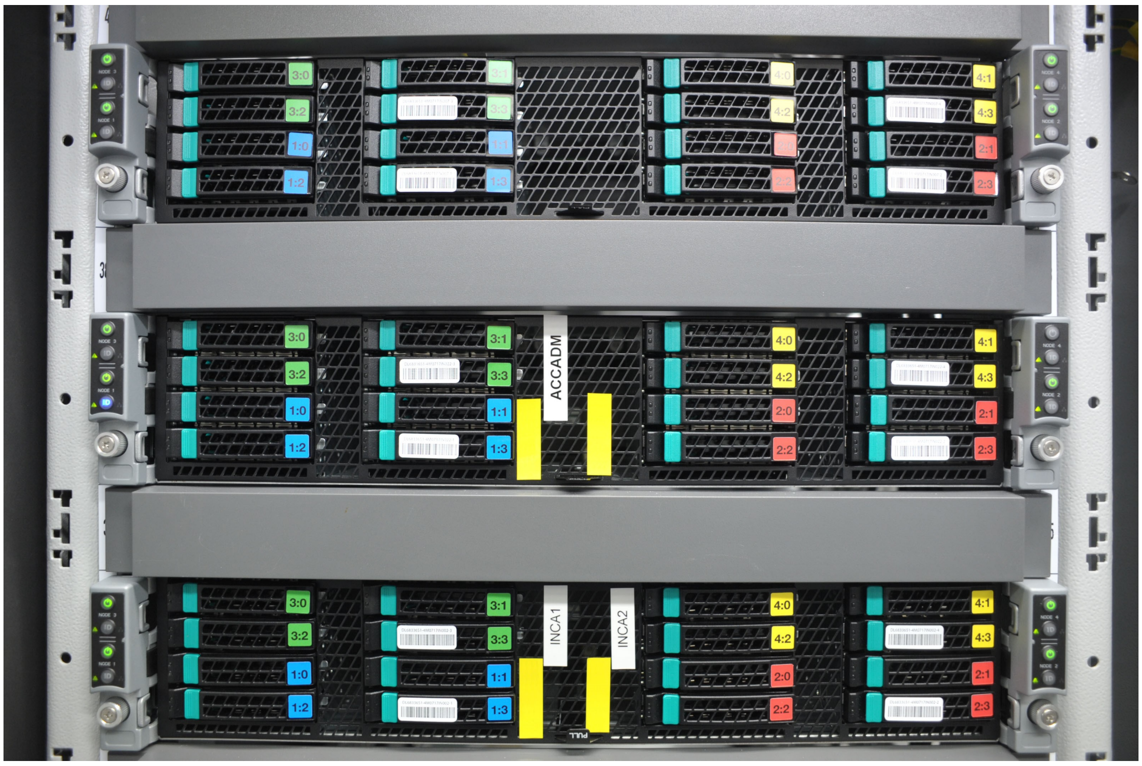


Figure 6.2: Quad servers

144TB. The JBOD enclosure provides redundant power supplies and a Serial Attached Small Computer System Interface (SCSI) (SAS) connection. On the server-side, the JBOD enclosure is connected through a Host Bus Adapter Card (HBA). The HBA allows a connection to 2 JBOD enclosures, but we currently do not require such large amounts of storage.

High-availability is vital, therefore, the enclosures are configured using software RAID 6, where 2 drives are dedicated to parity. An additional disk per enclosure is a hot spare. Since we rely on software RAID, the server connected to the JBOD is responsible for calculating the stripes and parity. Figure 6.3 shows a typical example of RAID 6 systems, where one can see the two drives dedicated to parity. This configuration allows us to survive up to 2 disk failures. This redundancy scheme was chosen because the recovery time is quite long and we wanted to ensure that redundancy is not lost during the reconstruction. Thanks to the hot spare, the array returns to full RAID without human intervention.

On top of the software RAID, we use Linux's Logical Volume Manager (LVM), which allows us to hide the physical volumes, thus providing flexibility for extensions.

Since the beginning of 2018, the NFS servers and NXCALs storage machines are based on JBODs. For NFS, this allowed the number of servers to be consolidated from 10 to 4 machines. We now have 172 quad servers in operation.

[9]By Colin M.L. Burnett [CC BY-SA 3.0], via Wikimedia Commons

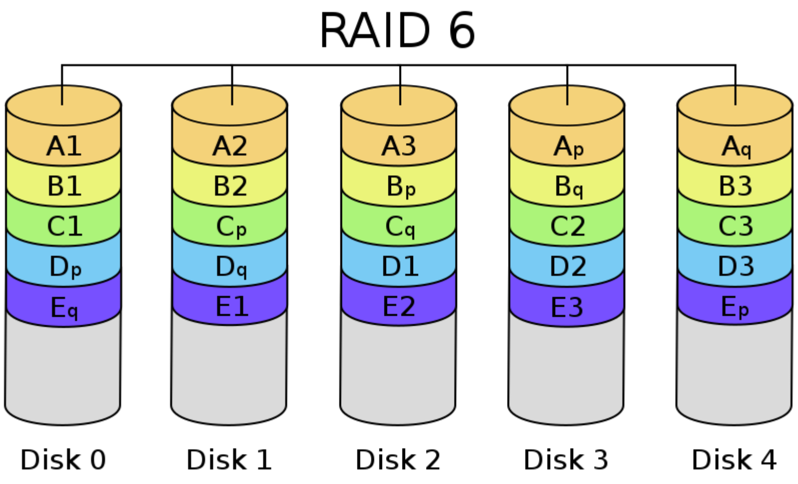


Figure 6.3: RAID 6 on 5 disks[9]

6.1.4 Installation and Administration

The server infrastructure is mainly installed in the CCR with just a few machines installed off-site for backup and fast disaster recovery purposes.

The racks used today are standard 19-inch wide racks. We have a mixture of HP racks and other generic racks, which are deeper, in order to handle future formats of quads. In cases where increased cooling is required, water-cooled external doors are fitted.

Each rack receives a double mains distribution; one from the Swiss network and one from the French network. In case of mains failure, Uninterruptable Power Supplies (UPSs) can provide power for one hour at full capacity. Afterwards, three diesel generators can take over the power generation.

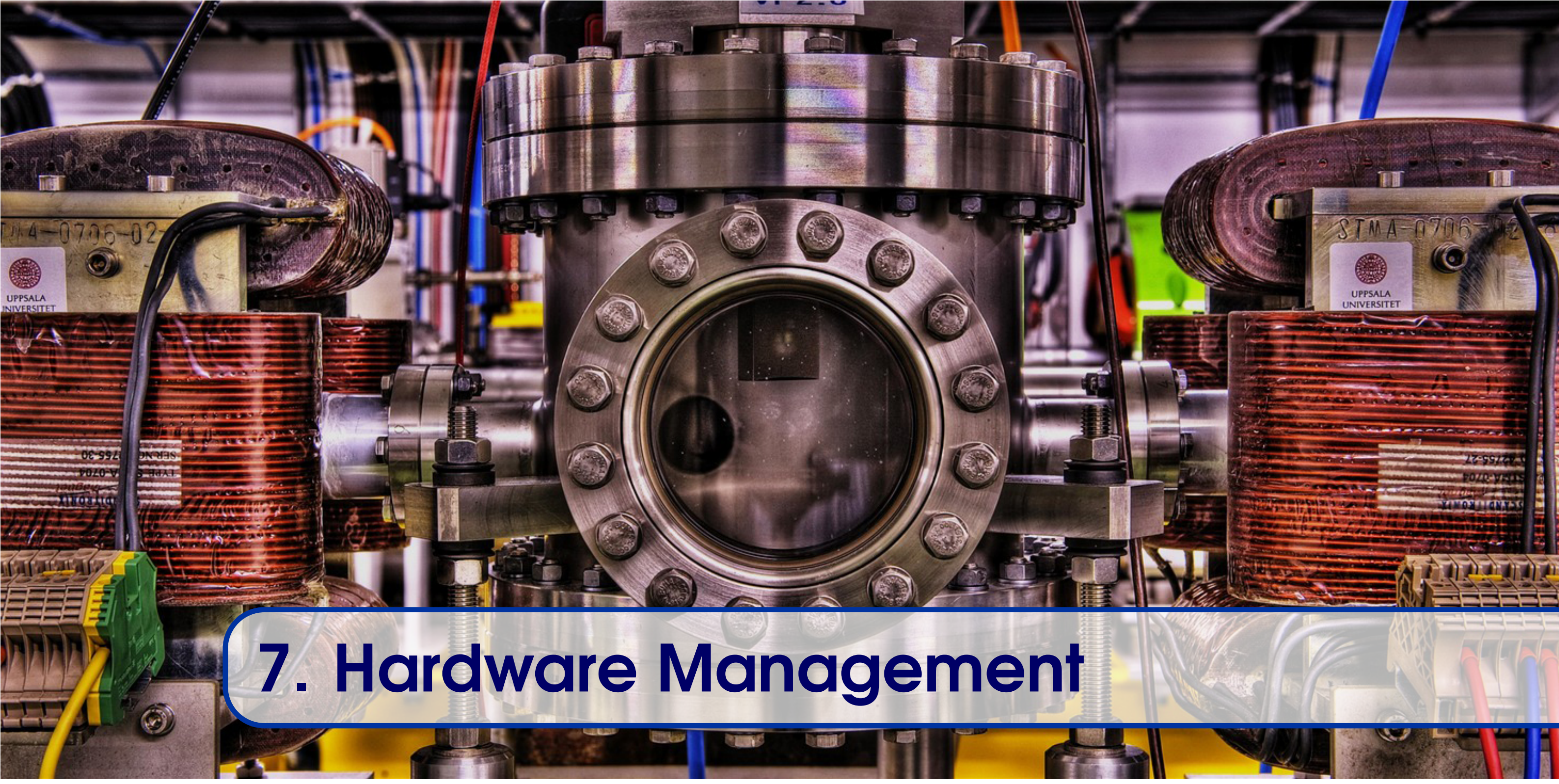
The operating system installed on most of the machines is the BE-CO Community Enterprise Operating System (CentOS) 7. In some cases, older versions such as Scientific Linux CERN (SLC) 6 are used. In the near future, CentOS 7 will be deployed on all the machines. Chapter 8 gives more details on the subject.

6.2 Consoles

For the consoles installed in the CCC, the local control rooms and the technical buildings, we use standard desktop PCs from the IT department. These PCs have Intel Active Management Technology (AMT) (similar to IPMI), which allows remote access and reboot. A typical configuration for the control rooms is a PC with three screens.

As more than 600 PCs are installed, to limit the number of interventions we require all of the components inside the PC to have a Mean Time Between Failure (MTBF) higher than 100’000 hours (approx. 10 years).

Most of the consoles run Linux (CentOS7 in 2019) but there are also a few Windows computers to run industrial applications.



7. Hardware Management

In the previous chapters we have described the hardware provided by BE-CO. Furthermore, BE-CO also provides a service to procure, install and repair this hardware.

Equipment groups request hardware installations by specifying their requirements through an online web portal. Before the installation can be performed, the surrounding infrastructure must be in place (e.g. cables, racks). Then, the hardware is prepared and asset management aspects are addressed. Specific pieces of hardware are taken out of the stock and physically installed. The last stage of the process is to bill the equipment groups for the hardware.

The first implementation of this process was based on a simple web form, which sent the request by email. It wasn't linked to any other data service, but it allowed the formalisation of the requests, departing from the old habit of receiving ad-hoc requests over the phone or email. The current system is based on Service Now (SNOW), a commercial service management portal that is heavily used by other services at CERN. This tool retrieves data from other sources, such as catalogue data from the asset management tool, Enterprise Asset Management System (InforEAM) and Racks from the Layout Database (see chapter 21). InforEAM is provided by the Computerised Maintenance Management System (CMMS) service from the Accelerator Coordination and Engineering (ACE) group in the EN department. SNOW provides a standard, user-friendly interface, suitable for many profiles of users. Other options widely used in the group, such as Jira, would not have offered the integration required.

Figure 7.1 depicts the hardware installation workflow, as implemented in SNOW, thanks to the help of the Service Management and Support group (SMB-SMS). As represented on the diagram, the installation request is also forwarded for information to the Machine Controls Coordinator (MCC) to allow them to follow the progress of the task. In Work Task 1 (WT1), the request is assigned to a member of the Hardware Installation team and analysed in order to identify the type and availability of the hardware required, as well as the need for any additional services such as cabling, networking etc. For the latter, BE-CO relies on other departments such as EN and IT. This may delay the installation, as cables or fibres may have to be laid. Once the infrastructure is fully present, the hardware is checked out of the stores in the asset management system, using the kiosk application and registered as part of an installation. Finally, the configuration is recorded in the Controls Configuration Database (CCDB) (see chapter 20).

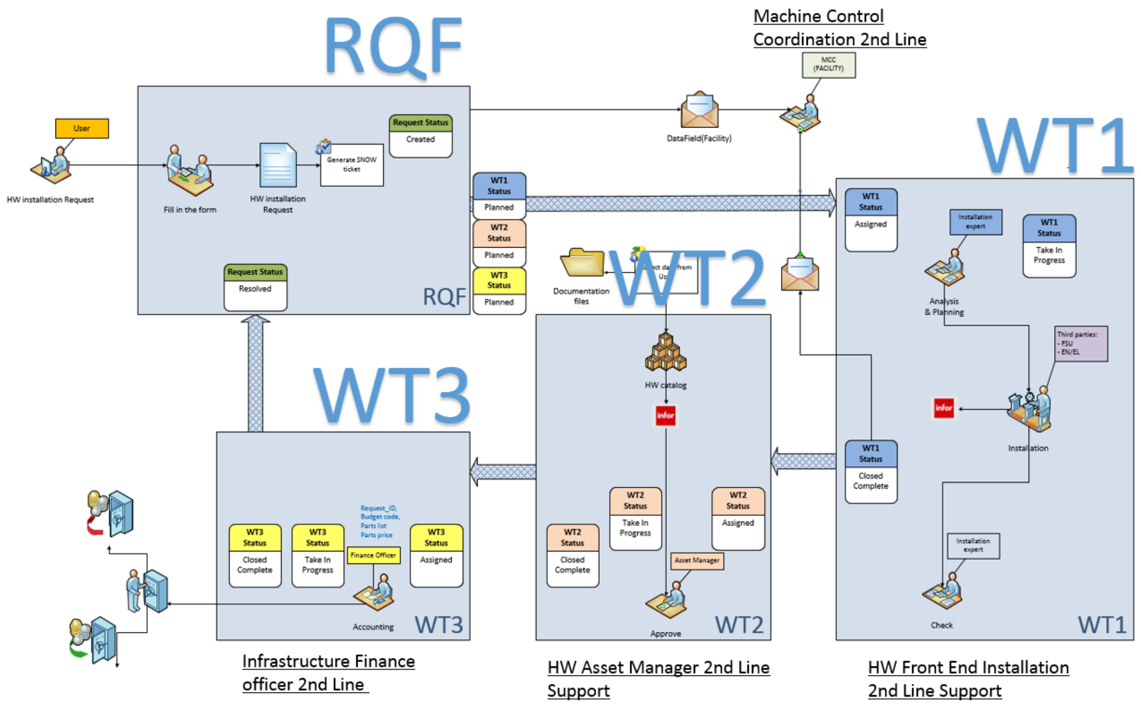


Figure 7.1: Hardware installation workflow

In the second phase (WT2), the Asset Manager takes over the request and ensures that the asset management has been handled properly in WT1. Indeed, there are several cases where the asset management cannot be done during WT1, such as when equipment is dismantled and put back into stock. In addition, the Asset Manager ensures that the stock levels are sufficient and triggers procurement if necessary. This role is important to ensure data quality and was particularly relevant during the early days when there were teething issues, for example a module which is in stock but is not registered in the database.

As nothing comes for free, the Infrastructure Finance Officer receives the hardware installation request in order to establish the invoice. At CERN, billing between groups is done through an Inter Departmental Transfer (TID) document in CERN's Electronic Document Handling System (EDH).

The decision to launch a procurement order depends on stock levels and forecasts based on consumption statistics and pending installation requests. For long shutdowns the PLAN tool is used to collect needs, in advance, from all equipment groups. Procurement is a difficult task for several reasons. Firstly, electronic components can have large lead-times, four to six months in some cases. Then, we have to deal with obsolescence policies from the manufacturers, where we are left with two choices; either to buy a supplementary quantity to cover our future needs, or to design a new module with modern components, the latter taking approximately 2 years. For example, in LS2 (2019-2020), there are several large consolidation projects happening in the group, such as the pulse repeater renovation, for which hundreds of new modules need to be procured and available before the shutdown begins.

Unfortunately, hardware modules sometimes break down and therefore we need to be organised in order to intervene efficiently, so that the accelerator's availability can be kept as high as possible. For CO systems, every accelerator has a dedicated hardware installation expert, who is called in case of hardware failure. After installation of non-CO systems, the equipment groups become the first-line operational support. They return broken modules to BE-CO for repair and hold a small stock of spare parts. Most of the time, the repairs are not billed, except for MEN CPUs and expensive modules such as fast digitisers. This policy works well to keep the machine downtime low, but adds another challenge to the asset management, as BE-CO is not always aware of changes and relies on being notified by the equipment groups when assets are replaced.

This process has been in place for several years and generally works well. However, the key challenge is data quality. Efforts are being made on several fronts to try to improve the situation. For example, new modules have serial numbers accessible online, allowing feedback loops between the databases and the installed hardware to be created (auto-discovery). An inventory of the stores should be performed periodically, the next on-site inventory will be done during LS2. Also, improvements to the integration between the different databases and tools, is being implemented under the Controls Hardware Data Management initiative, as described in figure 7.2. Finally, we want to be able to identify failure modes and ageing components, by defining failure types in EAM Light and assigning them to defective modules.

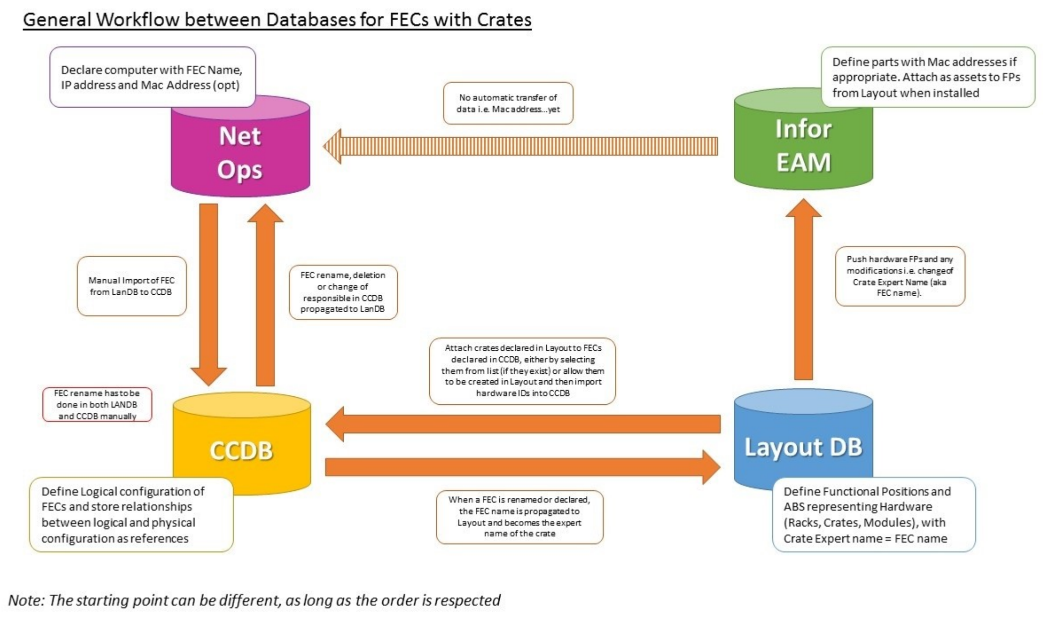

General Workflow between Databases for FECs with Crates
Declare computer with FEC Name, IP address and Mac Address (opt)
Net Ops
No automatic transfer of data i.e. Mac address...yet
Define parts with Mac addresses if appropriate. Attach as assets to FPs from Layout when installed
Infor EAM
Manual Import of FEC from LanDB to CCDB
FEC rename, deletion or change of responsible in CCDB propagated to LanDB
Push hardware FPs and any modifications i.e. change of Crate Expert Name (aka FEC name).
Attach crates declared in Layout to FECs declared in CCDB, either by selecting them from list (if they exist) or allow them to be created in Layout and then import hardware IDs into CCDB
FEC rename has to be done in both LANDB and CCDB manually
CCDB
Layout DB
Define Logical configuration of FECs and store relationships between logical and physical configuration as references
When a FEC is renamed or declared, the FEC name is propagated to Layout and becomes the expert name of the crate
Define Functional Positions and ABS representing Hardware (Racks, Crates, Modules), with Crate Expert name = FEC name
Note: The starting point can be different, as long as the order is respected

Figure 7.2: Generic workflow for Controls Hardware Data Management

8. Operating Systems

All of our software, except for the C code developed with MockTurtle, runs on top of an Operating System (OS). For the last 15 years, we have focused our efforts on a single choice: Linux. Of course, there are different requirements between the different types of hosts. The embedded Linux must have a reduced foot-print, the FECs' Linux must have better real-time characteristics than the standard kernel, and all of them must be very stable. The following sub-chapters present the various Linux configurations that we have, as well as some details on older OSs that are still in use and how we manage the OSs and their deployment on hundreds of computers.

8.1 Servers and Consoles

For many years, the servers and consoles have run under Linux. Most of these computers do not have real-time constraints and stability is the key aspect that should be maximised. Servers and consoles run CERN Community Enterprise Operating System (CentOS), which is a derivative of Red Hat Enterprise Linux (RHEL). From CentOS, CERN's IT department derives CERN CentOS, thus giving BE-CO the necessary upstream support. In each rebuild, custom packages are added, for example, CERN CentOS contains packages for user and printer management that are CERN-specific. As CentOS is based on Red Hat Enterprise Linux, we are guaranteed a version of Linux that will remain stable for 8 to 10 years, including the backport of bug fixes and the addition of new drivers to support modern hardware. In 2019, the version used in operation is CentOS 7 (64-bit) based on RHEL 7. The previous version of Linux for the consoles and servers was based on Scientific Linux (SL) developed at Fermilab. As shown in figure 8.1, both SL and CentOS are derivatives of RHEL. In order to further increase the stability of the server environment in the medium-term, we are investigating high-availability solutions that could be used for our critical services. The mechanism is based on an active/passive system, more specifically, a two-node configuration with one active and one passive node, which could be extended further. As open-source solutions are always preferred, pacemaker (a high-availability Cluster Resource Manager (CRM)) and corosync (a Group Communication System (GCS)) are used.

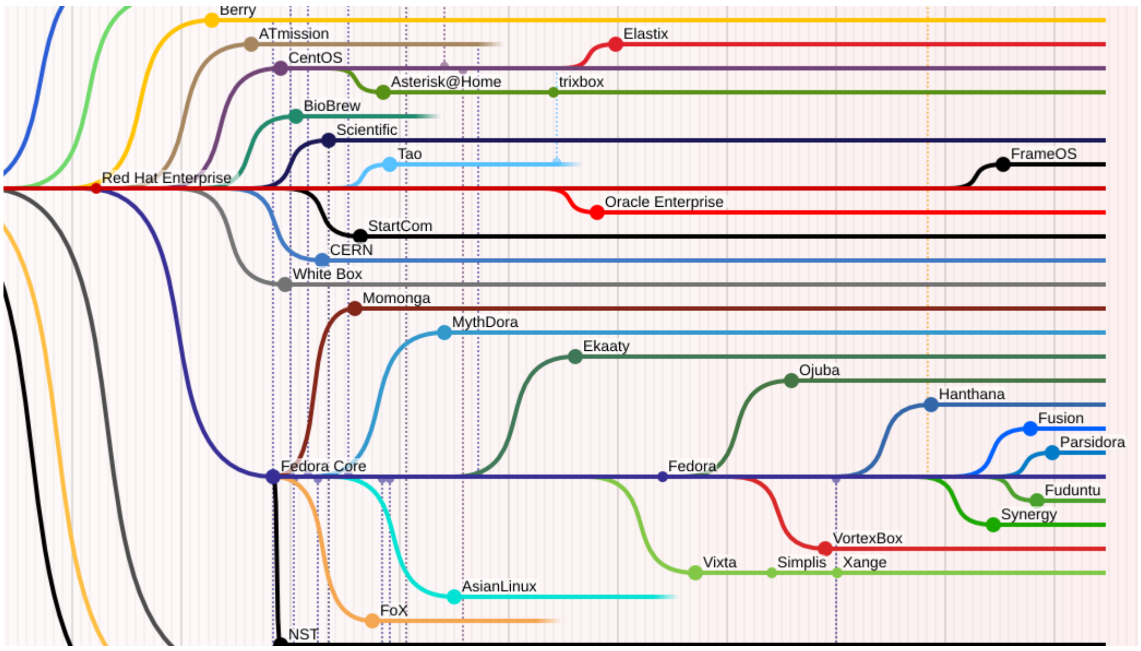


Figure 8.1: Extract of the Red Hat family tree focusing on RHEL

8.1.1 Configuration and Package Management

The Control System needs a number of specific packages in addition to the standard ones offered by IT. Moreover, in order to successfully manage the configuration and deployment of 600+ servers and consoles, we need to have our own repository and tools. We chose Ansible, a piece of Free Open Source Software (FOSS) from Red Hat, to perform the deployment and configuration management [10]. During the year, Ansible is used in pull-mode, where a Cron job pulls a Git branch and locally performs the installation and configuration. The Ansible push mode is used to perform checks, reboot hosts and perform further checks to ensure that they have returned to a good state. Additionally, it is used for general "orchestration" of hosts. For the process' execution and management, we still use a home-made tool called wreboot.

One of the key features of Ansible compared to other products such as Puppet, used by IT, is that Ansible does not require a custom agent to run on the to-be-configured machines. Instead, Ansible uses Secure Shell (SSH), which is available as standard on any Linux machine.

Ansible allows us to categorise different computers into groups and sub-groups by functionality (consoles, VirtualPCs, etc.) and assign roles to them. The roles define a set of tasks to be performed such as installation of packages, configuration of the machines, etc. Figure 8.2 illustrates the role hierarchy that we use for the development servers. For such a machine, in addition to the base set of packages, we add specific packages for the server environment and yet another set for the development servers.

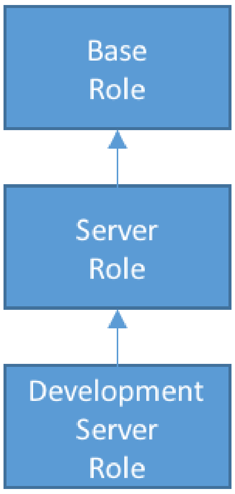


Figure 8.2: Roles given to a development server

8.2 Front-End Computers

The software running on the Front-End Computers (FEC) has to have real-time behaviour (see section 11.1 for more details on real-time software). In the early 90s, we chose LynxOS, a Unix-like, POSIX-compliant, hard real-time OS. The main reasons behind this choice were its support for multi-users and multi-processes, as well as its real-time features. Even though we are still using LynxOS on a few PowerPC-based operational FECs, we plan to replace them all with Linux-based systems by LS2.

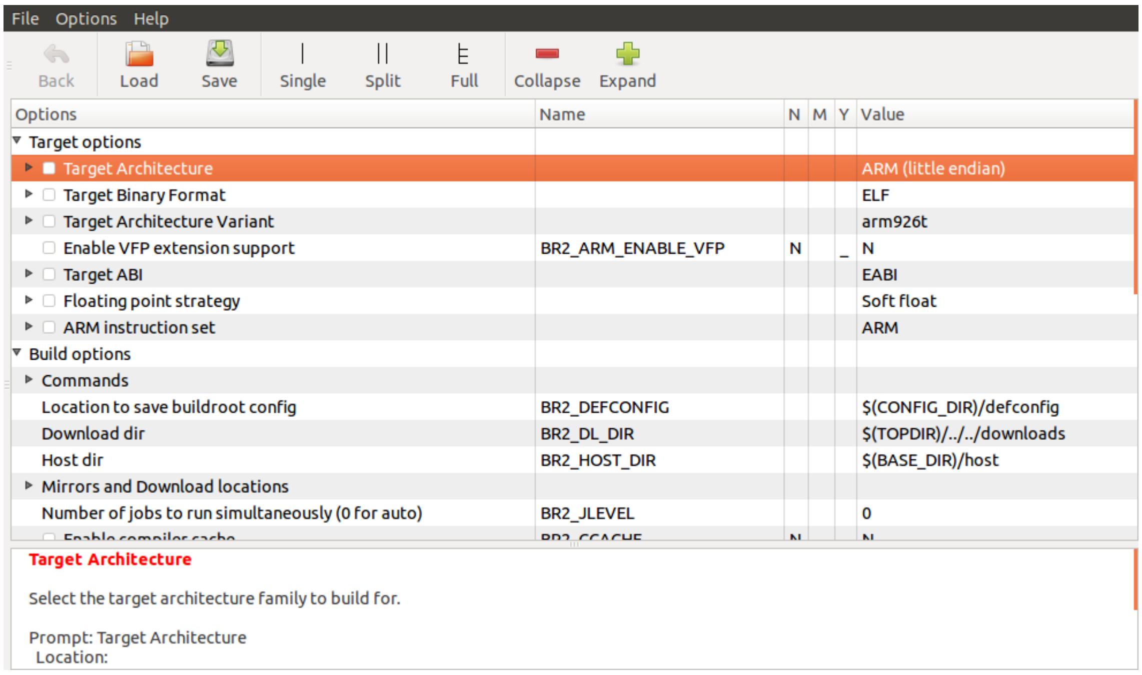
In the early 2000s, as the availability of Linux distributions increased, it was decided to use Linux on the FECs, as well as on the consoles, as this allows us to better homogenise the OS landscape of the Control System. As the servers and consoles are based on CERN CentOS 7 (CC7), it is logical to use the same distribution for the FECs. In 2019, we support three versions of Linux for front-end computers.

On one hand, we have the SLC versions, a 32-bit SLC5 (L865), based on RHEL5, and a 64-bit SLC6 (L866), based on RHEL6. On the other hand, the next-generation of Linux for the FECs, which is available since the end of 2017, is based on CentOS 7 (see section 8.1) and is 64-bit with a real-time kernel. Unfortunately, when we started to use SLC5, there were no readily available real-time kernel packages so we had to configure and compile our own kernel to fit our needs. With SLC6, there is a Messaging Real-time and Grid (MRG) repository where a real-time kernel package is available. This kernel provides low-latency responses to events, something only required in our real-time applications, and is therefore applied to the FECs but not typically to the servers and consoles.

As explained in chapter 5, the FECs are diskless machines and the OS must be downloaded into a RAM disk at boot time. From the IT Dynamic Host Configuration Protocol (DHCP) server and our boot servers, the booting FEC obtains all necessary information required to download the OS image and proceed with its startup. This mechanism is based on PXE Boot, a combination of DHCP and Trivial File Transfer Protocol (TFTP).

8.3 White Rabbit Switches

For the White Rabbit switches we need a simple and efficient embedded Linux image. The White Rabbit switches have limited space both in terms of mass storage (512MB of flash) and RAM (64MB). Therefore, we need to limit the number of tools installed in order to minimise the footprint of our distribution. Thanks to the switch's architecture, the ARM CPU is limited only to administrative tasks and does not have any real-time constraints. Hence, the kernel built for the switches does not need any specificities, such as latency guarantees, unlike the front-ends. Unfortunately, due to maintenance complexity and the high number of tools available by default, we cannot reuse neither the SLC nor CentOS distributions. Instead, we decided to use a tool, BuildRoot, to generate our own tailored-to-our-needs distribution. Figure 8.3 depicts the GUI that can be used to customise our Linux distribution.



Options	Name	N	M	Y	Value
▼ Target options					
▶ Target Architecture					ARM (little endian)
▶ Target Binary Format					ELF
▶ Target Architecture Variant					arm926t
Enable VFP extension support	BR2_ARM_ENABLE_VFP	N		_	N
▶ Target ABI					EABI
▶ Floating point strategy					Soft float
▶ ARM instruction set					ARM
▼ Build options					
▶ Commands					
Location to save buildroot config	BR2_DEFCONFIG				$(CONFIG_DIR)/defconfig
Download dir	BR2_DL_DIR				$(TOPDIR)/../../downloads
Host dir	BR2_HOST_DIR				$(BASE_DIR)/host
▶ Mirrors and Download locations					
Number of jobs to run simultaneously (0 for auto)	BR2_JLEVEL				0
Enable compiler cache	BR2_CCACHE	N			N

Target Architecture

Select the target architecture family to build for.

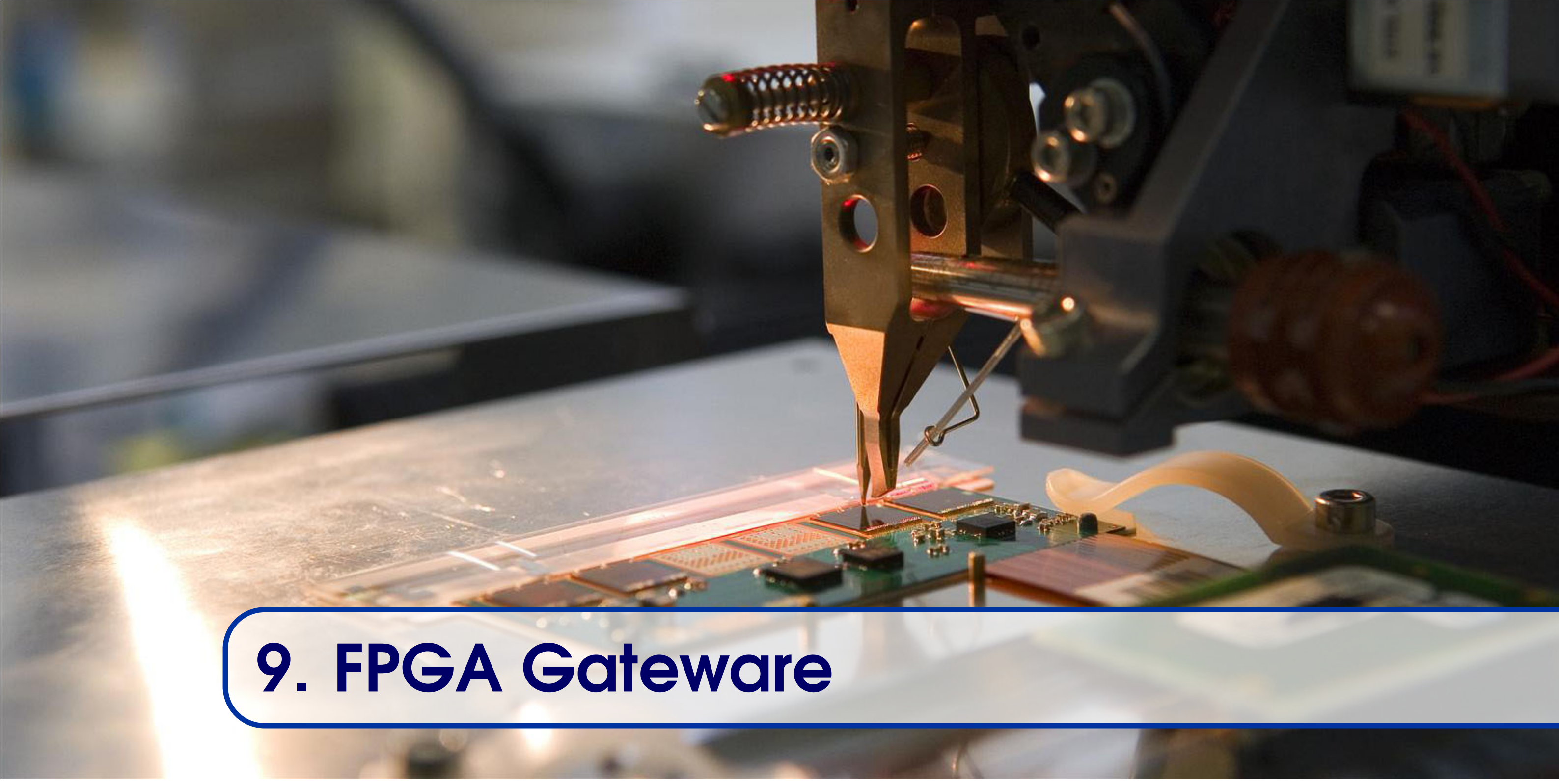
Prompt: Target Architecture
Location:

Figure 8.3: BuildRoot's GUI to select the options of the WhiteRabbit switches' OS

For the management of the switches (updates, etc.), we do not currently use any tools such as Ansible. A manual procedure is followed twice a year, or more often if a patch must be applied urgently. Even if the procedure is still manual, the updates can be performed remotely. Currently, a simple configuration file stored on NFS is used but the ideal solution would be to store the required information in the CCDB and generate the file automatically.

III

Front-End Software



9. FPGA Gateware

Since the 90s, most of the electronic boards use an FPGA on the PCB to implement the digital logic. The FPGA can be seen as a sea of logic gates and flip-flops that have to be configured in order to produce a useful result. This configuration is achieved by downloading a binary bit-stream into the FPGA. As with software, the development is not done at a low-level but instead using Hardware Description Languages (HDLs). The two main HDLs are VHDL and Verilog. Once the HDL code is written, the code is translated into a configuration bit-stream in a two-stage operation (synthesis and place and route). The HDL code is commonly referred to as gateware and the same engineering principles apply to gateware as they do to software.

One of these principles is modularisation. Features such as memory access or external bus access (e.g. VME bus controller) are implemented in separate modules to properly structure the gateware and ensure good reusability. Those modules are referred to as HDL cores. In some cases, the HDL cores are also known as IP cores, where the IP stands for "Intellectual Property".

To interconnect the different cores, an internal bus is required. In 2012, it was decided to use the Wishbone Bus, as it was the only free, open-source option available at this time. From 2019 onwards, another bus, Advanced eXtensible Interface 4 (AXI4), will also be supported. Figure 9.1 gives an overview of the FPGA architecture used to support the 100 MSa/s 4-channel FMC ADC when plugged into the SPEC carrier.

The main advantage of such an architecture is that it becomes relatively easy to exchange blocks connected to the internal bus. For example, the PCIe core (top-right corner of figure 9.1) can be replaced by a VME bus core.

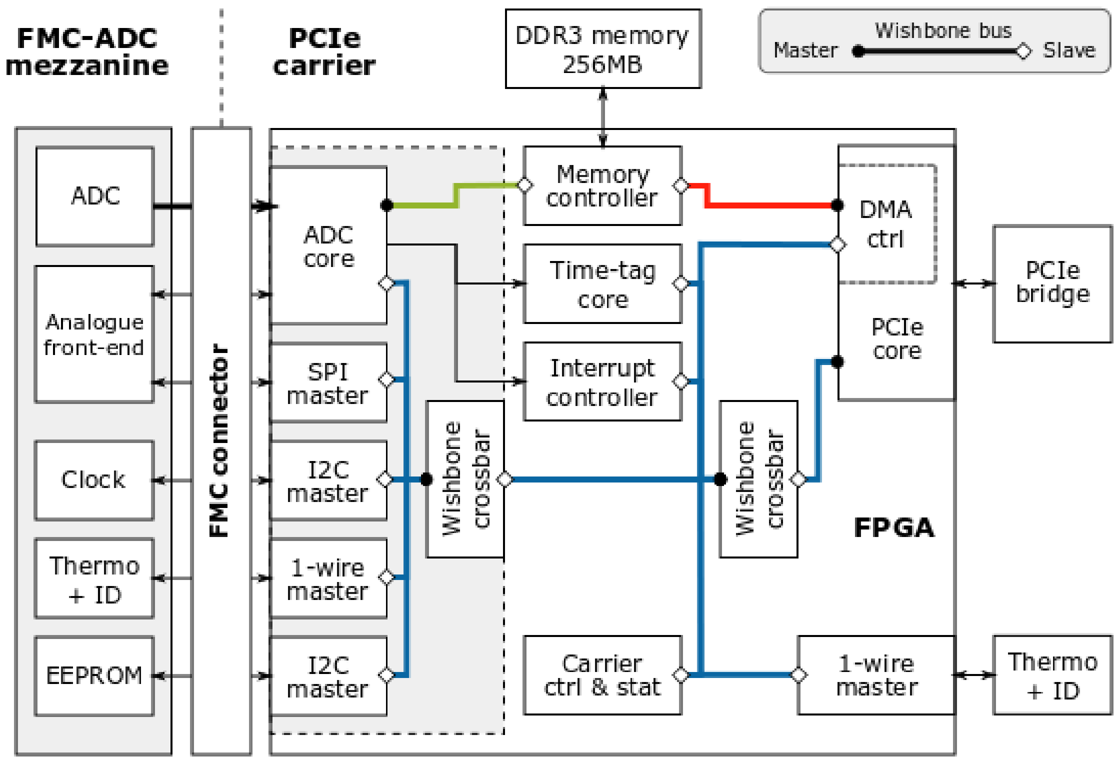


Figure 9.1: Gateware architecture of the 100MSa/s 4-channel FMC ADC

In 2019, the BE-CO group provides, among other components, cores for:

- Bus controllers (VME, PCI, PCIe, etc.)
- Memory access
- Wishbone Bus interconnect
- White Rabbit/PTP core

Nevertheless, the HDL development remains a complex task that can be lengthy. Even though simulation tools exist, the iteration time between versions is much longer than one typically finds in software development. To alleviate this problem, the group provides two tools, HDLMake and Wishbone Generator (wbgen) to facilitate and streamline the development process. In addition, in order to broaden the range of profiles able to work with FPGAs, the MockTurtle core has been developed.

9.1 HDL Development Tools

HDLMake facilitates the HDL development work by coordinating the many tools normally required. The input to HDLMake is a set of design files, used to call the synthesiser, the place and route, simulator, etc. In 2019, the tool supports the two main FPGA brands, Xilinx and Intel (formerly Altera).

For the interconnection of the cores, another tool called wbgen is used. For example, in order to connect two cores, the first step is to describe the registers that each HDL core exposes, and which need to be connected to the internal bus. From this description, wbgen generates the bus interfaces. The logic of the cores is implemented directly in HDL by the developer, as well as the connections between the bus signals of the two cores. For cases where more cores are involved, an additional Wishbone Bus crossbar core is required, but the rest of the process remains the same.

In the future, HDLMake will probably be replaced by FuseSoC, an open-source, third-party solution, and wbgen will be integrated directly into Cheby (see section 12.1).

9.2 Mock Turtle

MockTurtle is an FPGA core and software framework that facilitates development of hard real-time applications [62]. The main goal of MockTurtle is to reduce development time and allow C programmers to develop in the FPGA without knowledge of any HDL. This is achieved by providing an HDL module implementing small CPUs, known as soft processor cores or soft cores, that can be programmed in bare-metal C. Figure 9.2 depicts MockTurtle's architecture with its main external connections.

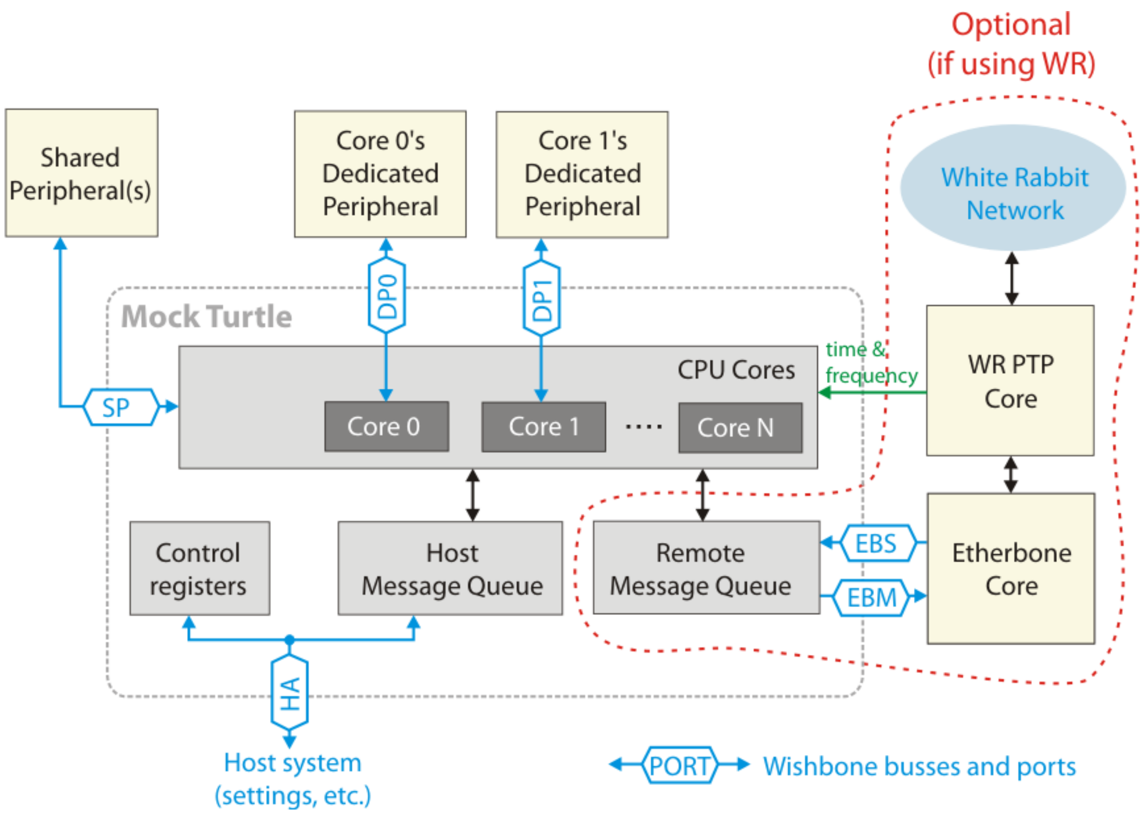
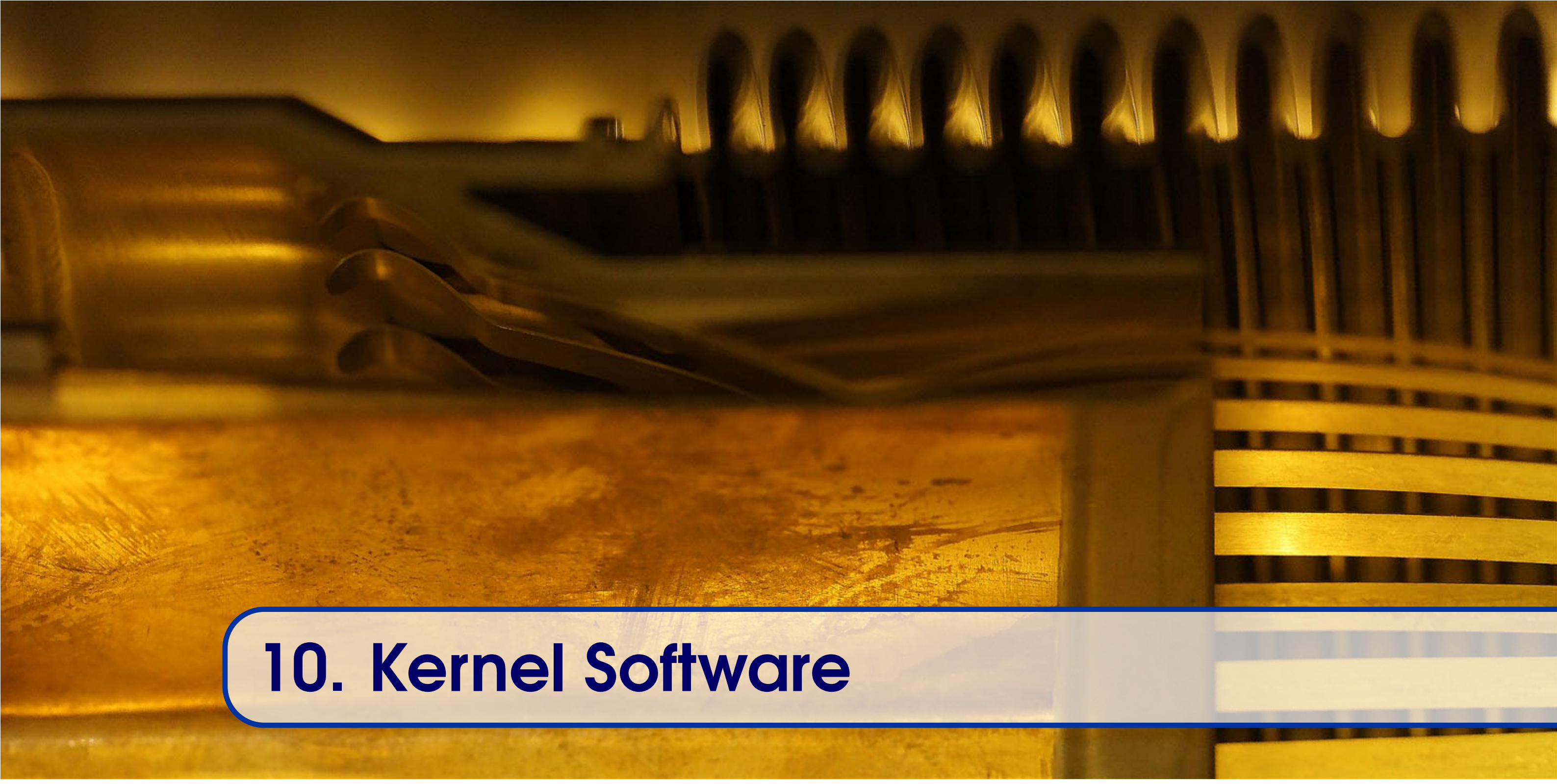


Figure 9.2: Mock Turtle's architecture

MockTurtle is configurable and can support up to eight CPUs whenever computations have to be performed in parallel. MockTurtle also provides a means to synchronise the different CPUs (semaphores) and access shared variables. Initially, the CPU implemented was the LM32, a 32-bit microprocessor design from Lattice Semiconductor. In the future, the LM32 will be replaced by the CERN-designed micro Reduced Instruction Set Computer (RISC) 5 (μRV), a simple CPU implementing the RISC V instruction set available with a Lesser General Public Licence (LGPL). The C code written for the MockTurtle CPU is simply compiled using the GNU's Not Unix (GNU) C Compiler (GCC)[10] toolchain and GNU Debugger (GDB) support is foreseen. Whenever some logic would be better implemented in hardware, it is easy to add logic blocks. To interface with the external world (Ethernet, VMEbus, etc.) a uniform system of First In First Out (FIFO) is provided. Of course, a UTC input port fed by White Rabbit is optionally available.

In 2019, the product is still under development but the plans are to bring it to a level so that it can be offered as a BE-CO service. In the future, a possible evolution would be to close the gap between a pure bare-metal C environment and a real-time framework such as FESA (see section 11.2 for more details on FESA).

[10]recursive acronym GNU's Not Unix (GNU)



10. Kernel Software

Whenever our application software needs to access a piece of hardware, it does so by relying on the operating system's kernel. The Linux kernel can be seen as a big monolithic piece of code that gets compiled and linked in one go; all of the variables have to be resolved at compile/link time. Nevertheless, the kernel accepts extensions through kernel modules. Kernel drivers (aka device drivers or simply drivers) are the pieces of kernel code that handle I/O operations and interrupt handling. Most of the time, the drivers are built as kernel modules. The drivers expose their services to the applications through system calls such as ioclt, read write, etc. Another alternative is to ask the kernel to map I/O addresses in the process's addressable memory and to handle the interrupts directly in the user-space code. Unfortunately, this setup is very inefficient as it involves a lot of expensive context switching between the user space and the kernel space. Another advantage of using a kernel driver is that it allows for easier concurrency management as the locking can be done in a single place; the kernel code. For those reasons, we systematically use drivers to access our front-end computer's hardware. As drivers expose limited operations to user-space (ioctl, read, write, select...), the driver is typically accompanied by a user-space library that provides a clean C API. In the rest of this chapter, and unless specified otherwise, whenever we write driver we mean the kernel module and its user-space library. Figure 10.1 depicts the different spaces in a Linux system as well as the location of the typical components.

The driver is the first opportunity to add a level of abstraction to the Control System. The driver should provide an application-agnostic functional interface that hides the actual implementation details of the hardware modules they control. The application software (e.g. a FESA class) will then use that library's API to implement specific applications using the hardware modules. Modifications to the implementation details will not be seen by the applications, as they are shielded by the driver. For some very common hardware (e.g. a serial line such as an RS232 port), the interface has been standardised further and, in the example of a serial line, one expects to see the hardware through a TTY type interface.

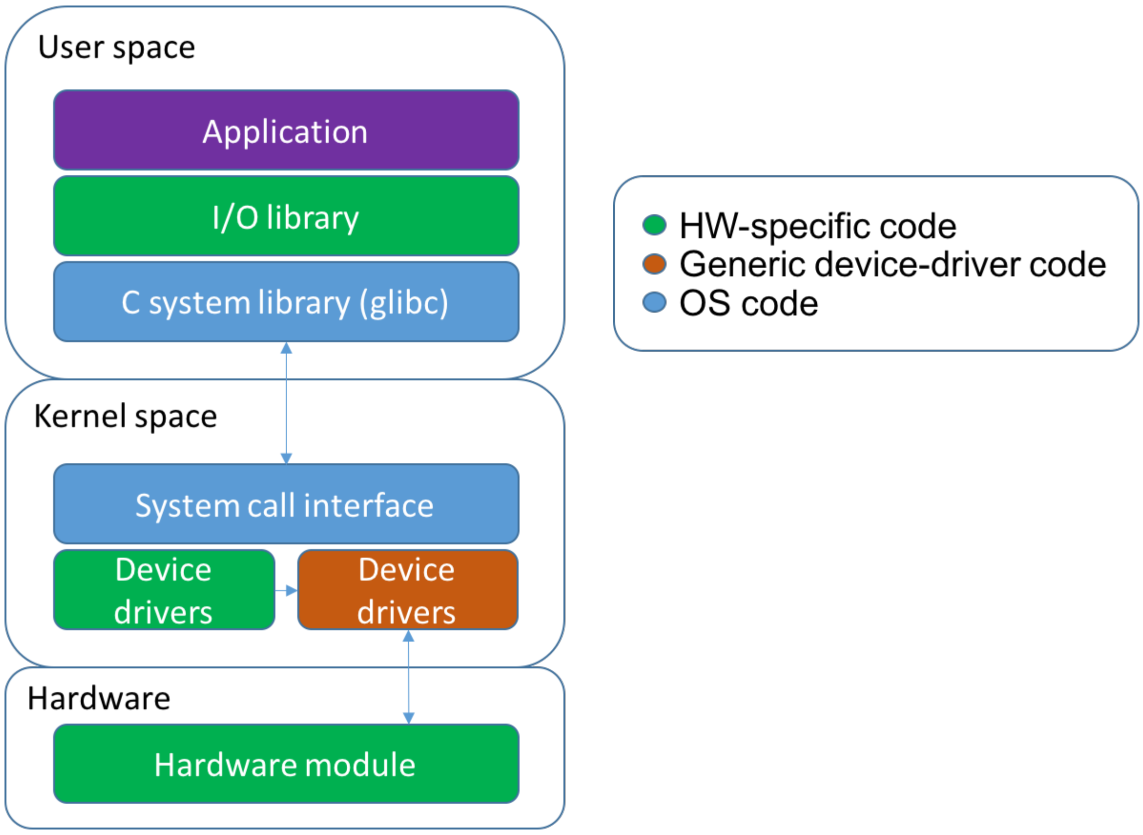


Figure 10.1: User and kernel spaces in Linux

The developer of a new board driver can rely on kernel services, and the resulting modules require symbols from the kernel that are resolved during the module installation (`insmod`). In turn, the installed module can export other services (i.e. symbols) that then can be used by other modules. A good example of this structure is the VME driver that provides an in-kernel API that can be used directly by the device drivers by relying on symbols exported by the VME driver.

While the Application Binary Interface (ABI) is very stable, the in-kernel APIs are much more subject to refactoring between versions of the kernel. This means that, whenever a new kernel is supported by the group, an effort has to be made to adapt the drivers to the new kernel. Modern kernels evolve less than they used to (e.g. from 2.6.14 to 3.6.11) but this is an important point to keep in mind when planning the development of drivers.

For all of the module types in the BE-CO hardware kit, the group provides a driver with its user-space library and test program. This set is available for all of the operating systems currently supported by the group although it is possible that the newest modules are only available for the newest OS's. In addition, we provide raw VME bus access thanks to a driver interfacing with the PCI-to-VME bridge; the Tundra TSI148 chip on the MEN A20. Access to PCI/PCIe buses is already fulfilled by the Linux kernel and therefore we don't need to provide a custom driver. As explained in section 5.4.3, we try to systematically open-source our hardware developments [15, 57]. In turn, we can share the device driver giving access to the module. Instead of having a specific public repository and to have to install the driver manually, one can upstream the driver and get it integrated into the kernel directly. This is something that the BE-CO group has done a few times, but the additional time required to properly upstream a device driver means that we can only afford to do it for very generic drivers or modules. Furthermore, while this makes perfect sense from a Linux community contribution point-of-view, since we use very stable versions of Linux, there is a long delay between our contribution being included in the kernel and being able to profit from it. A typical case for a device driver that we might want to upstream is the driver we will write for the new MEN A25's VME bridge, which is based on an FPGA and an open-source HDL design.

When it comes to system configuration, we want to automatically install the drivers when a FEC starts, depending on the hardware modules present in the enclosure. The Controls Configuration Database (CCDB - see chapter 20 for more details) is used to store drivers' configuration and the modules installed in a given FEC. When the FEC is prepared (aka generated), the transfer.ref file is generated and the driver installation commands with all of the parameters are included. Because the VME configuration requires a lot of parameters to be given, we do not currently use Linux standard mechanisms. In the future, the release location of the drivers and the way we install them should be reviewed aiming towards a more Linux-standard approach.

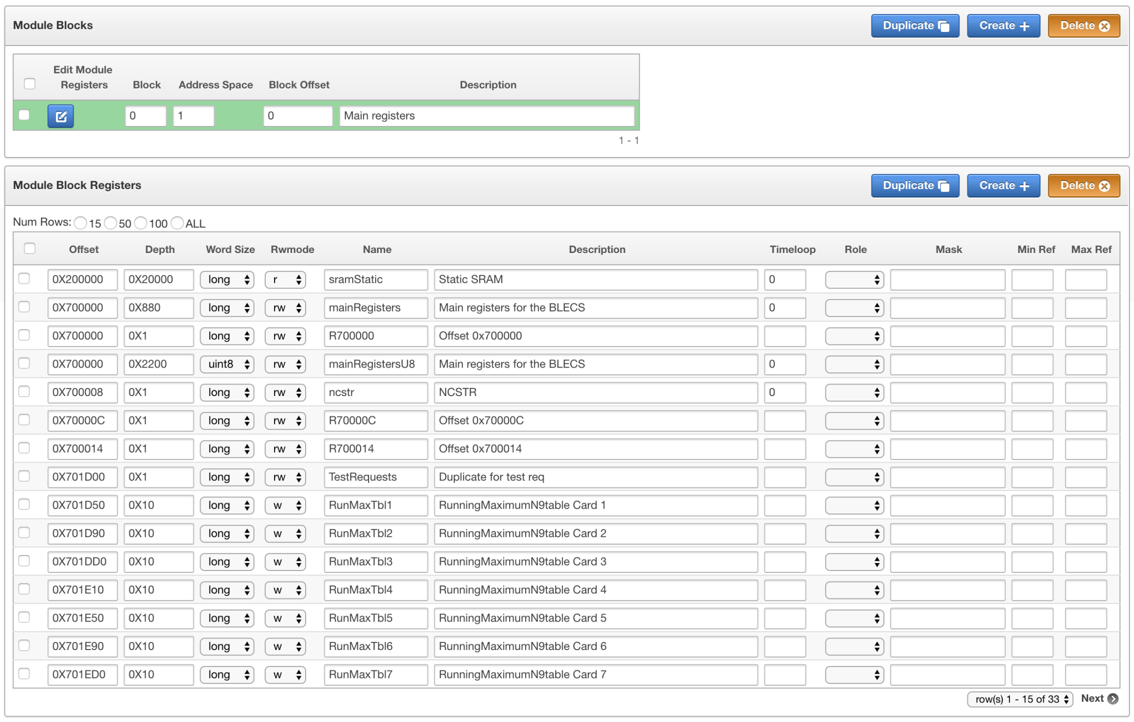
10.1 Kernel Software Development Tools

Developing kernel code is very difficult, as almost any bug introduced by a driver will stop the computer and require a complete reset. For that reason, BE-CO provides two tools, Encore and Encore Driver GEnerator (EDGE), to generate device drivers and their user-space libraries from descriptions.

10.1.1 Encore

Encore generation is based on a VME module description (addresses, access mode, memory size, etc.) from the Controls Configuration Database (CCDB). In addition to the device driver and its user-space library, Encore also produces a Python-based test program. This can be used to create small Python expert applications to test the hardware. With a tool such as Encore, the user does not provide any kernel code and the stability of the system is ensured by having only well tested code in the kernel space. Even the interrupt handling is done in the user space but some fine tuning of the interrupt handling kernel code is still possible, for example, change the interrupt queue size. Figure 10.2 is a screen-shot of the Encore's register definition page.

Unfortunately, the API that the library exposes is a simple mapping of the different registers (either a narrow or a wide interface). By relying on such a tool, we miss an opportunity to increase the level of abstraction and the driver is just there to handle the transition between the access-limited user space and the kernel. The consequences are that the logic that should be in the driver is now in the application (e.g. a FESA class) making it more complex than necessary and vulnerable to hardware implementation details changes. Also, the overall performance is impacted as a context switch between user space and kernel has to be done for each and every register operation.



Edit Module Registers	Block	Address Space	Block Offset	Description
	0	1	0	Main registers

1 - 1

Num Rows: 15 50 100 ALL

Offset	Depth	Word Size	Rwmode	Name	Description	Timeloop	Role	Mask	Min Ref	Max Ref
0X200000	0X20000	long	r	sramStatic	Static SRAM	0				
0X700000	0X880	long	rw	mainRegisters	Main registers for the BLECS	0				
0X700000	0X1	long	rw	R700000	Offset 0x700000					
0X700000	0X2200	uint8	rw	mainRegistersU8	Main registers for the BLECS	0				
0X700008	0X1	long	rw	ncstr	NCSTR	0				
0X70000C	0X1	long	rw	R70000C	Offset 0x70000C					
0X700014	0X1	long	rw	R700014	Offset 0x700014					
0X701D00	0X1	long	rw	TestRequests	Duplicate for test req					
0X701D50	0X10	long	w	RunMaxTbl1	RunningMaximumN9table Card 1					
0X701D90	0X10	long	w	RunMaxTbl2	RunningMaximumN9table Card 2					
0X701DD0	0X10	long	w	RunMaxTbl3	RunningMaximumN9table Card 3					
0X701E10	0X10	long	w	RunMaxTbl4	RunningMaximumN9table Card 4					
0X701E50	0X10	long	w	RunMaxTbl5	RunningMaximumN9table Card 5					
0X701E90	0X10	long	w	RunMaxTbl6	RunningMaximumN9table Card 6					
0X701ED0	0X10	long	w	RunMaxTbl7	RunningMaximumN9table Card 7					

row(s) 1 - 15 of 33 Next

Figure 10.2: Register definition in Encore

10.1.2 EDGE

Building on the functionality provided by Encore, EDGE adds PCI support, versioning, simulation, and a better hardware description format. VME was for many years the format of choice for the design of electronic boards, but with more and more designs using the PCI and PCIe buses, a tool to generate device drivers for those buses was required. One of the challenges of supporting PCI is the heterogeneity of the bridge implementations. Indeed, EDGE provides read/write PCI access but generic support of the interrupt handling and the DMA transfer is not possible, as it was with VME. Nevertheless, EDGE brings a unified treatment of the DMA and interrupt handling, thanks to its plug-in architecture. EDGE currently supports the Gennum 4124 and the Xilinx PCI Express DMA IP core bridges. Support for more bridge implementations will be added when the need arises.

The development of EDGE was also an opportunity to improve the user-space library's API and provide a long-awaited feature: hardware simulation. In 2019, the latter is still quite basic, but based on the text file, it is now possible to simulate the response of the hardware module. EDGE provides two APIs, the first is generic, while the second is specific, based on the hardware description. Similarly to Encore, EDGE also generates a test program, as described in section 19.2.1.

The workflow of EDGE, as shown in figure 10.3, does not differ substantially from Encore's process.

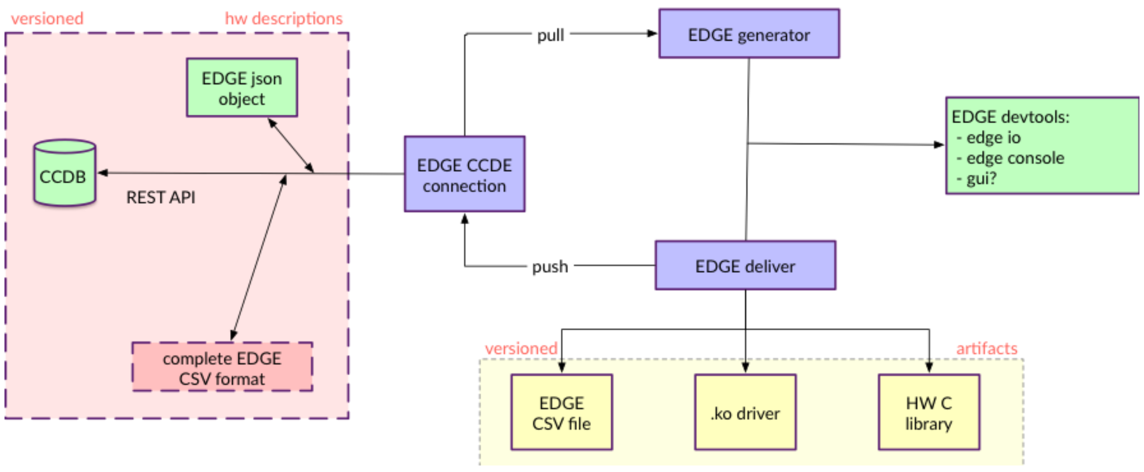


Figure 10.3: EDGE’s workflow

11. FEC Applications

The front-end computer (FEC) applications have two main roles. Firstly, they perform the low-level control of the hardware, and secondly, they expose the low-level variables to the external world, for the high-level Control System to access. In the accelerators' Control System, we choose to expose these variables using a paradigm known as the device-property model. The part of the FEC applications that implements the device-property is commonly referred to as the device server.

As explained in the previous chapter, the accelerator equipment's reference values (aka settings) change continuously depending on the beam being produced. The vast majority of the hardware modules used in the Control System are not aware of this setting multiplexing and are controlled by a single set of settings at a given time, without having any knowledge of the accelerator's timing. Therefore, the FEC applications need to perform the control of the hardware following the events and instructions (user, destination, etc.) distributed by the timing system. In addition, as the operators need to monitor the beam production and the state of the accelerators continuously, the FEC software sends a constant flow of hardware acquisitions to the high-level Control System. A piece of software that reacts to events and processes them before a deadline is commonly referred to as real-time software. Parts of the FEC applications are designed to fulfil these real-time constraints. They are known as real-time systems.

11.1 Real Time Systems

When we talk about a real-time system, we must qualify the hardness of the real-time requirements. We should also ensure the real-time hardness of the solution and how it matches the requirements. There are three adjectives commonly used when describing a real-time system; hard, firm and soft. When talking about real-time requirements, the adjective depends on the consequences of missing a deadline and the usefulness of the result when it occurs after the deadline. In a hard real-time system, missing the deadline is a total system failure, while in a firm real-time system this situation is tolerable but degrades the quality of service. In both cases, the usefulness of the result drops to 0 as soon as the deadline is missed. In a soft real-time system, the usefulness of the result decreases after the deadline (100% at the deadline) until it reaches zero. Figure 11.1 depicts the usefulness of the result before and after the deadline for hard/firm (left) and soft (right) real-time systems. Similarly, a real-time solution offers deadline guarantees that can be hard or soft.

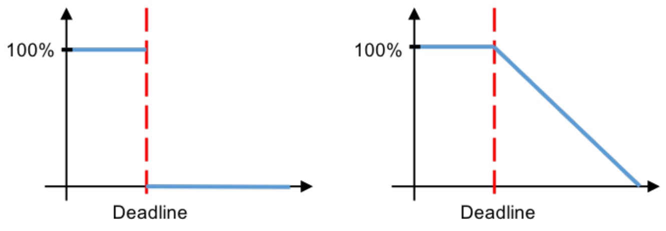


Figure 11.1: Degradation of the processing's usefulness in hard/firm and soft real-time systems

When qualifying the real-time FEC applications and the CERN accelerators they control, we have either hard or firm requirements depending on the piece of equipment at hand. For example, if we look at the constraints put on the control of a power converter feeding a bending magnet, it is clear that the usefulness of controlling the current falls to zero after the deadline; setting the right current in the magnet after the beam has passed is useless. The consequences depend very much on the element and where it is in the accelerator chain. A bending magnet in the transfer line between the AD machine and one of its experiments will only degrade the quality of service i.e. the amount of beam provided to the experiment's physicists, whilst the same type of element in the LHC might actually destroy the accelerator and should be considered as a hard real-time constraint. Some of the high-level applications have soft real-time constraints.

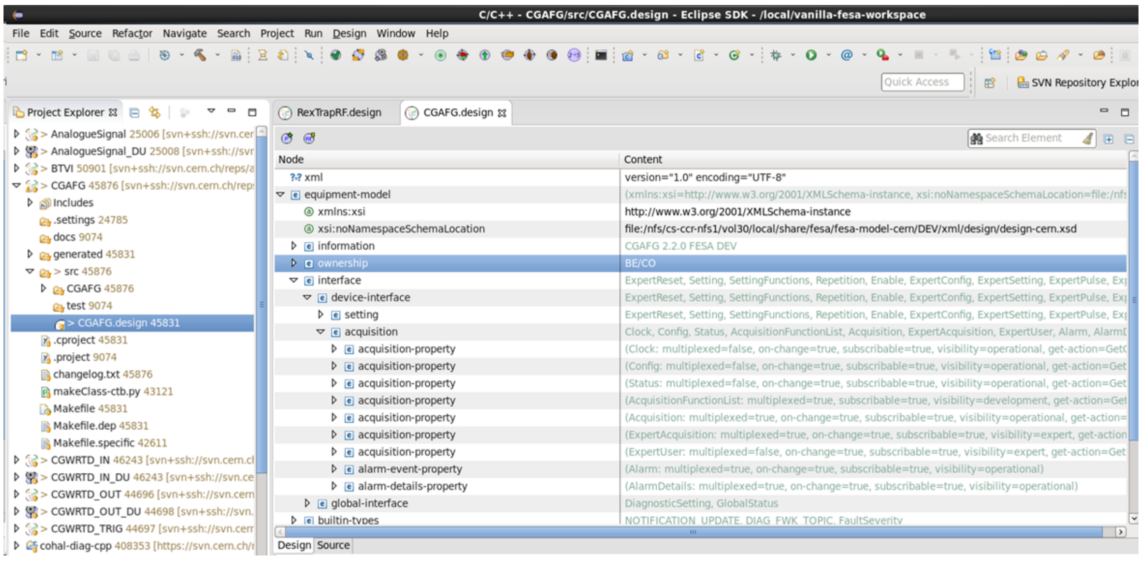
FEC applications must be designed and implemented with real-time constraints as well as respecting the structure and behaviour of the device-property model, which is not necessarily straightforward. In order to ensure better integration with the higher layers, the BE-CO group provides a real-time framework that allows equipment experts to design and implement their real-time applications, including the device server part. This allows the optimisation of the resources required for such a development. The latest version of this framework is called FESA3 (Front-End Software Architecture).

11.2 FESA

FESA, Front-End Software Architecture, is the real-time application framework to be used for any real-time application and device server developments in the accelerator Control System [5]. In 2019, the latest version of FESA is FESA3 V7.2.0. FESA offers developers a set of features to solve common problems in a common way, bringing a CERN-wide approach to the low-level software development. The main aspects covered by the framework are:

- Device server modelling (device-property model)
- Automatically generated get and set methods for properties
- Data-consistency checks for incoming and outgoing values (property get/set)
- Internal data support (device fields)
  - Initialisation, persistence and restoration
  - Data-consistency in multi-threaded, multi-core environment [32]
- Real-time behaviour modelling
- Real-time event dispatching with low jitter
- Configuration-driven real-time activity threading
- Support for alarms and run-time metrics
- Source code versioning, release and deployment

By developing an application with the FESA framework (called a FESA class), developers not only benefit from ready-made solutions to many of their problems, but they are also guided through the process. Furthermore, the resulting piece of software has a common structure, making long-term support and maintenance easier. The FESA development environment is based on an Eclipse plug-in, shown in figure 11.2, which allows the user to perform all the steps (modelling, coding, compilation, test, release) in a single Integrated Development Environment (IDE).



```
> AnalogueSignal 25006 [svn+ssh://svn.cer
> AnalogueSignal_DU 25008 [svn+ssh://svr
> BTVI 50901 [svn+ssh://svn.cern.ch/reps/a
> CGAFG 45876 [svn+ssh://svn.cern.ch/rep:
    Includes
    .settings 24785
    docs 9074
    generated 45831
    > src 45876
        CGAFG 45876
        test 9074
        > CGAFG.design 45831
    .cproject 45831
    .project 9074
    changelog.txt 45876
    makeClass-ctb.py 43121
    Makefile 45831
    Makefile.dep 45831
    Makefile.specific 42611
> CGWRTD_IN 46243 [svn+ssh://svn.cern.cl
> CGWRTD_IN_DU 46243 [svn+ssh://svn.ce
> CGWRTD_OUT 44696 [svn+ssh://svn.cern
> CGWRTD_OUT_DU 44698 [svn+ssh://svn.
> CGWRTD_TRIG 44697 [svn+ssh://svn.cerr
cohal-diag-cpp 408353 [https://svn.cern.ch/r
```

Node	Content
?-? xml	version="1.0" encoding="UTF-8"
equipment-model	(xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance, xsi:noNamespaceSchemaLocation=file:/nfs
xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
xsi:noNamespaceSchemaLocation	file:/nfs/cs-ccr-nfs1/vol30/local/share/fesa/fesa-model-cern/DEV/xml/design/design-cern.xsd
information	CGAFG 2.2.0 FESA DEV
ownership	BE/CO
interface	ExpertReset, Setting, SettingFunctions, Repetition, Enable, ExpertConfig, ExpertSetting, ExpertPulse, Ex
device-interface	ExpertReset, Setting, SettingFunctions, Repetition, Enable, ExpertConfig, ExpertSetting, ExpertPulse, Ex
setting	ExpertReset, Setting, SettingFunctions, Repetition, Enable, ExpertConfig, ExpertSetting, ExpertPulse, Ex
acquisition	Clock, Config, Status, AcquisitionFunctionList, Acquisition, ExpertAcquisition, ExpertUser, Alarm, AlarmD
acquisition-property	(Clock: multiplexed=false, on-change=true, subscribable=true, visibility=operational, get-action=GetC
acquisition-property	(Config: multiplexed=false, on-change=true, subscribable=true, visibility=operational, get-action=Get
acquisition-property	(Status: multiplexed=false, on-change=true, subscribable=true, visibility=operational, get-action=Get
acquisition-property	(AcquisitionFunctionList: multiplexed=true, subscribable=true, visibility=development, get-action=Get
acquisition-property	(Acquisition: multiplexed=true, on-change=true, subscribable=true, visibility=operational, get-action=
acquisition-property	(ExpertAcquisition: multiplexed=true, on-change=true, subscribable=true, visibility=expert, get-action
acquisition-property	(ExpertUser: multiplexed=false, on-change=true, subscribable=true, visibility=expert, get-action=Get
alarm-event-property	(Alarm: multiplexed=true, on-change=true, subscribable=true, visibility=operational)
alarm-details-property	(AlarmDetails: multiplexed=true, on-change=true, subscribable=true, visibility=operational)
global-interface	DiagnosticSetting, GlobalStatus
builtin-types	NOTIFICATION_UPDATE, DIAG_FWK_TOPIC, FaultSeverity

Figure 11.2: Design view in the FESA Eclipse plug-in

The user code is kept to a minimum and there are only three entry points where custom code can be injected. The first point is the so-called specificInit methods that are called to initialise the system (software and hardware). The two other points are the server actions and real-time actions. Server actions are classes that can be implemented by the developer when specific code needs to be executed on a get or set call. Developers can also choose to use the default implementation provided by the framework, which performs basic boundary checks and read-write from/to internal device fields. Real-time actions are C++ classes triggered by real-time events and executed in a threading environment, depending on the FESA class design. The code path between the event reception and the action execution is guaranteed to be deterministic, lock-free, and wait-free. Care is taken to avoid jitter-generating constructs such as dynamic-memory allocations. The real-time actions are the typical location for placing the code that manipulates the hardware. Both action types have a limited, easy-to-use environment with direct access to the operation context (selector, event) and the devices implicated in the operation; a single device for a server action, or a device collection for a real-time action.

The basic workflow for the development of a FESA class is as follows. First, one has to describe the properties and their value-items (name, type, dimensions, etc.), declare custom types such as enumerations, and define an internal device data model (fields). For the real-time part, one declares the events that the software must react to (timer, timing, custom events) and how those events must be handled (how many threads, how to distribute the devices on the threads, etc.). Finally, one declares how the updates must be propagated to inform the high-level Control System of the new values. Once the modelling is done, the C++ code can be generated. In most of the cases, only the real-time actions remain to be implemented, as this is where the hardware is controlled and this requires equipment-specific code. With the actions implemented, the developers can compile and test their software. If the software has to be deployed on operational FECs, it must first be released, following a strict version control.

Most of the steps described above can be done in an isolated environment, making FESA suitable for off-line development or even development outside CERN [43, 58]. Nevertheless, the FESA class description is sent to the CCDB at release time, in order to make the class interface (properties and custom types) available to the high-level Control System.

11.3 Other Real-Time Frameworks

There are other frameworks than FESA3 and while most of them are obsolete with well-defined end-of-life dates, one, called FGCd, is fully supported and used by the TE-EPC group for the control of the Function Generator Controller (FGC)-based power converters [35]. For the sake of completeness, we mention the names of the previous frameworks: FESA2, General Module (GM) and SLEquip, the latter being actually a device server framework rather than a real-time application framework.

11.4 Generic FESA Classes

As CO provides the hardware kit to cover generic hardware functions, the group also supplies the drivers, the low-level test program and the FESA classes. This allows the modules to be integrated into the Control System and used "out of the box", without requiring further development. With these building blocks, users can develop a controls solution to a problem quickly and with little investment.

The generic FESA classes are grouped in a family called Controls Generic Front End Software (CGFES). Their core functions are:

- Timing, such as fine delay (CGTFDEL) and time-to-digital converters (CGTDC)
- Function generation for arbitrary (CGAFG) and periodic (CGPFG) functions
- Basic digital (CGDIO) and analogue I/O (CGAI, CGAO)

To reduce the maintenance effort, the CGFES FESA classes all rely on the CO Hardware Abstraction Library (COHAL). This abstraction layer allows us to integrate new generations of modules with equivalent functionality into the FESA class. In order to reduce the support cost of declaring new instances of CGFES, we plan to develop a better instantiation tool that would allow the users to be more independent.

12. Low-Level Development

In part III, the chapters describe the different layers of software that BE-CO provide to the equipment groups so that they can build their specific low-level applications. These applications, deployed on top of controls hardware as described in chapter 5, perform the control and acquisition of the accelerator equipment.

To recap, a typical low-level controls application is comprised of an electronics board, which has an FPGA for which some gateware will be required (see chapter 9). In order to interface the board with the RT application (see chapter 11), a kernel driver is needed (see chapter 10). It must be noted that sometimes, the electronics board is installed remotely. In this case, different hardware solutions such as remote I/O and fieldbuses can be used, as described in sections 1.5 and 5.5, combined with a software component such as Software Infrastructure for Low-Level Equipment ControllerS (SILECS) to interface with the RT application (see section 12.2).

Requiring several components to build a specific controls application can be cumbersome as every iteration requires the developer to go through all of the layers. In order to ensure a rapid and efficient development process, the development tools need to be properly integrated. The next chapter describes Cheby, which aims to orchestrate the different tools described above.

12.1 Cheby

Cheby was developed to ease the development of hardware-software interfaces, thus avoiding repetitive and tedious work. One of the main goals is to have a single source for the description of the interface, which allows the generation of several artefacts such as HDL, drivers, etc. Cheby is an integration tool in the sense that it provides the common description, but it relies on other specialised tools to generate code. Figure 12.1 depicts the integration of Cheby with other front-end software tools to generate HDL, drivers, and FESA class designs.

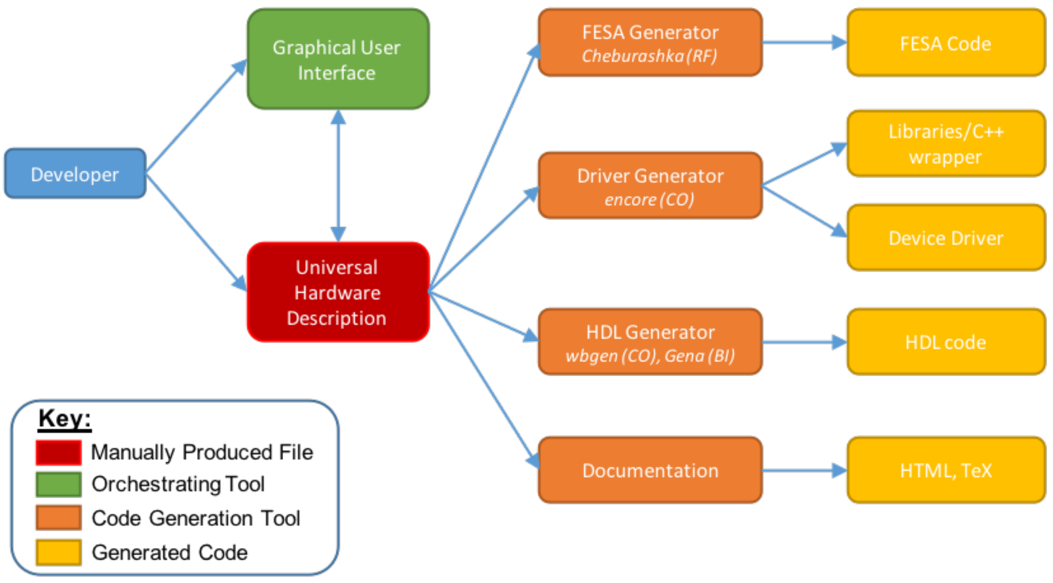


Figure 12.1: Integration of Cheby with other front-end software tools

The hardware-software interface, also known as a memory map, is described as a YAML text file. The main concepts applied in the file are registers with bit-fields and RAM. In order to structure the interface description, registers and RAM are logically grouped into blocks. Furthermore, interface descriptions can be included in other descriptions as sub-memory maps, in order to allow the reuse of existing mappings. For example, the description of a commonly-used HDL core can be directly included in the description of an electronic board.

Given an interface description, Cheby delegates the generation of the wishbone hardware interface to the wbgen tool (see section 9.1). It relies on Encore or EDGE to generate the driver and the user library and finally calls Cheburashka to generate a basic FESA class design. Optionally, Cheby can generate a simple C structure representing the software interface for cases where the device driver is developed manually.

It should be noted that by relying on tools such as Cheby for the development of low-level controls, many abstraction layers are bypassed. Ideally, once the final design is obtained, some effort is required to abstract low-level details in the appropriate layer. For example, the device driver should hide the hardware implementation details from the RT layer and the RT software, FESA in most cases, should expose an API with a higher level of abstraction to the upper layers of the Control System, instead of registers and bit-fields.

The first production-ready version of Cheby was released in late 2018. The tool is only available with a command-line interface, which might be seen as a drawback to some users, but offers scripting capabilities. In the future, more options will be available, such as the choice between Wishbone Bus and AXI4 for the hardware interface. Further integration in the Control System is also foreseen, by storing the Cheby descriptions in the CCDB.

12.2 SILECS

As commercial Supervisory Control and Data Acquisition (SCADA) solutions such as PVSS and WinCC OA are not well-integrated in the BE-CO Control System, we needed a layer to ease the communication with the industrial components. Software Infrastructure for Low-Level Equipment ControllerS (SILECS) is a tool to streamline the data exchange between clients (e.g. FESA classes) and low-level controllers that are typically unable to integrate with the FEC-based Control System. Originally, SILECS, then called Ethernet Interconnection for Programmable Logic Controllers (IEPLC) [41], was used only for PLCs, as they are neither powerful nor open enough to run Linux and FESA. Nowadays, the number of controllers SILECS can support has grown and, in addition to industrial solutions from Siemens, Schneider, and Beckoff, SILECS can also communicate with mini-PCs, LabVIEW PXIe crates and even FPGAs.

In addition to hiding the different communication protocols such as Siemens's S7, Modbus, or LabVIEW shared variables, the SILECS layer defines a data model based on data categories, configuration, command, and acquisition. Influenced by the device-property model, this approach of data block exchange also offers better performance. This common model has the advantage that it is only at deployment time that the target controller must be specified. Furthermore, this approach makes it much easier to look at others' designs as the structure is common.

The approach taken by SILECS is very similar to the one used in FESA. Based on the user design, code is generated in order to reduce or even eliminate the need to write code. Depending on the target object, SILECS generates either code or a data mapping to be uploaded to the low-level controller and a configuration file to customise the generic client library.

The user design is produced using a stand-alone GUI based on the Eclipse Rich Client Platform (RCP). This has the advantage of not requiring an operational Eclipse installation, whilst retaining the possibility to use the Extensible Markup Language (XML) validation, Apache Subversion (SVN) integration, etc. provided by Eclipse. This is convenient, as in this case, there is no C++ coding/compilation required. A possible future evolution is to provide an Eclipse plug-in contribution that could be integrated with the FESA plug-in.

The first step of the SILECS workflow is to define the data elements being exchanged with the controller. The design is stored in an XML file and must be validated against an XML schema defining the SILECS meta-model. In the second step, the tool is used to generate both the controller specific code (C code, mapping, LabVIEW configuration) and the XML configuration file that the client library reads to perform the read-write access. Several points are worth noting with this design. It is as un-intrusive as possible, as on most of the targets, there is no server or anything else to be run. There is a clear separation between the data structure design step and the rest of the process which makes the necessary skill set smaller, as the user does not need to be knowledgeable in both PLC and FESA. The client library, which can be used from any C++ application, is very generic. An additional step allows the generation of a design-specific wrapper library in order to further ease the copying between the low-level controller and a FESA device (direct read/write from/to the FESA device's fields). Figure 12.2 summarises the SILECS workflow.

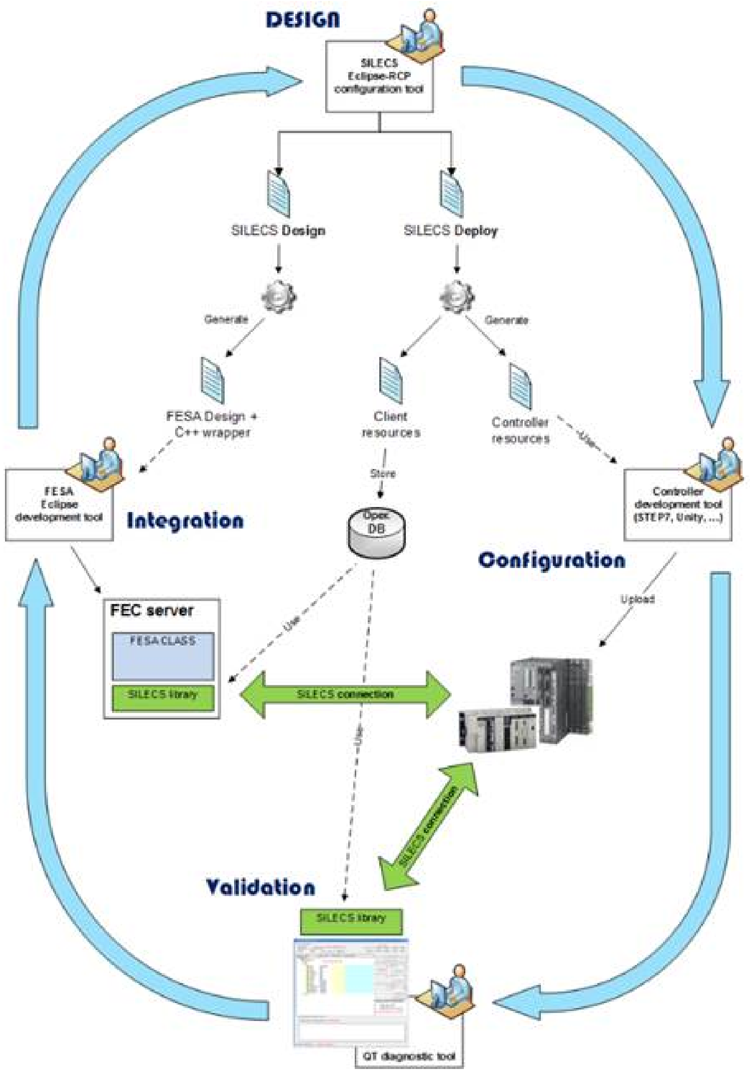


Figure 12.2: SILECS workflow

As visible on the figure, there is a generic diagnostic tool that can be used to validate and diagnose a system. This tool has been developed using C++ and Qt and uses the very same client library as the final client application.

With the arrival of more and more Ethernet-based solutions, one could ask what is the rationale to use SILECS rather than directly run a FESA class on Linux. The FEC infrastructure is based on either SLC or CentOS CERN and, in 2019, FESA is only available for the Intel CPUs. Many controllers will not have enough memory or CPU power to run the full stack properly and many of those are based on ARM CPUs that are currently (2019) not supported. In addition, some of the smaller controllers (so-called Ethernet couplers) do not have any programmable logic; they only give access to their I/Os. Going a step further, prototypes have been made to exchange data blocks between FECs and FPGA directly without a host on the FPGA side. This is based on the Wishbone Bus architecture in the FPGA and Etherbone that connects the internal Wishbone Bus to the Ethernet.

IV

Communications

13. Networking

13.1 Ethernet Networks

The networks for the controls components are based on Ethernet. As one can expect in such a large organisation, there are several Ethernet networks at CERN, but as far as the Control System is concerned, we only have to work with two networks; the General Purpose Network (GPN) and the Technical Network (TN). The GPN is dedicated to public hosts with Internet access while the TN is reserved for equipment used to operate the accelerators and the technical infrastructure. A detailed description of the GPN is outside the scope of this document and it is sufficient to know that most of the development laboratories' equipment is installed on the GPN.

The TN has a backbone with a star topology but, for reliability reasons, there are some interconnections between the leaf nodes. Each leaf also acts as a local star point i.e. at the level of the technical building. For example, the CCR is a star point to which all of our servers are connected. In 2019, the backbone is based on a 10GB Ethernet infrastructure. The TN does not have any direct Internet connectivity. Nevertheless, some controlled communication is possible between the two networks for both interactive sessions and services [29].

The front-end computers, introduced in chapter 5, are connected to the network using either one or two sockets. For the hosts supporting Intel's AMT, a single connection is sufficient for both the normal traffic and the host diagnostics. In short, AMT uses the same physical link as the computer, but a separate chip intercepts the traffic and responds to specific commands. This architecture allows AMT to work even if the main computer is OFF as long as the chassis is powered. Typical AMT-enabled hosts are the industrial PCs based on PCI or PCIe, but also the more recent PXIe CPUs. For the open enclosures such as VME, the standard setup requires two connections, one for the normal traffic on the SBC, and a second one to the enclosure's fan tray. This allows the monitoring, through SNMP, of several important metrics (e.g. temperatures, fan speed, etc.) and also the capability to perform actions directly on the crate, such as switching the system ON and OFF.

For the backend servers (see section 6.1 for more details), the network connections either use the specific switches included in the HP ProLiant enclosures (2019), or external switches provided in IT racks. For the HP solution, a network switch is located in each enclosure and the connection is shared between all of the servers within. This approach is very practical for installation and maintenance, but one has to keep in mind that the bandwidth is then shared. For the next generation of servers, the quad enclosures follow a slightly different model, with the switches installed directly in a central rack dedicated to network equipment. There are two connections; The first connection provides TN connectivity and the second is used for the Intelligent Platform Management Interface (IPMI) link. IPMI is another management and monitoring solution for computer systems. IPMI is more complex than AMT and the main physical difference is that it uses out-of-band management as opposed to AMT that applies in-band management i.e. using the same physical link.

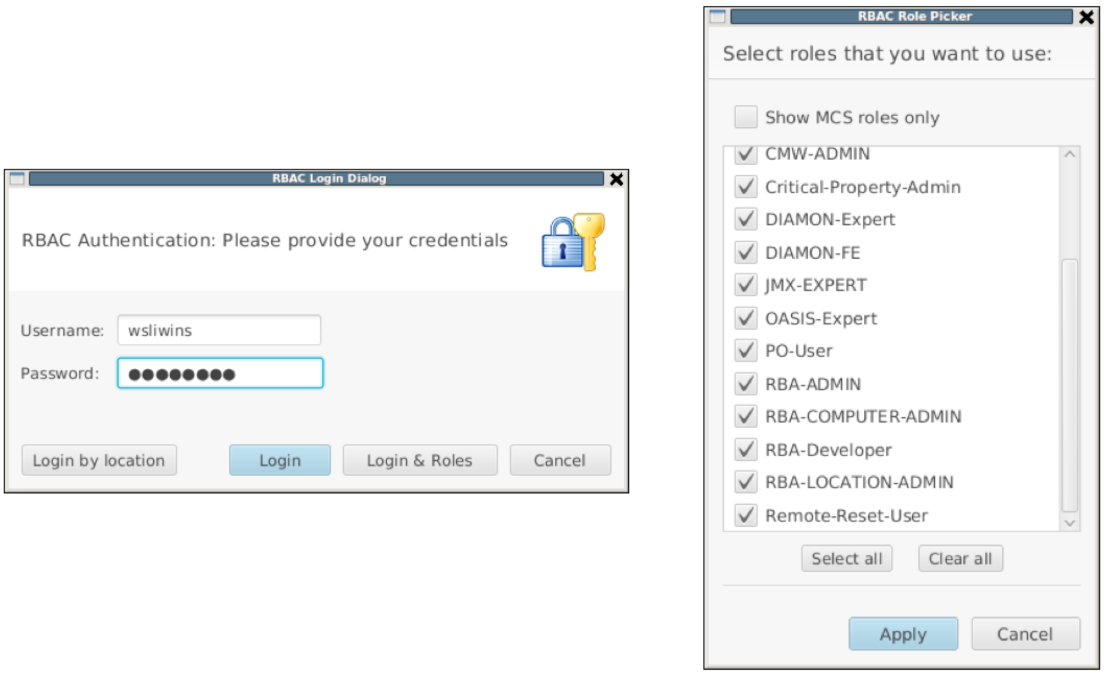
Over time, the number of connected devices will continue to grow and therefore the network capabilities must increase in line with this expansion. Recently, powerful application-specific computers, such as the ObsBox (Observation Box) from the BE-RF group, needed to be connected and required dedicated fibres.

13.2 RBAC

Role-Based Access Control (RBAC) is a component to protect access to certain resources in the Control System. RBAC's main purpose is to protect properties of selected devices. It must be stressed that Role-Based Access Control (RBAC) is by no means designed to protect against malicious users, but to prevent someone from performing the wrong action at the wrong time which, during LHC operation, could cause the loss of the beam.

RBAC grants access to resources using roles and rules. A rule stipulates that a resource is only accessible, in a combination of read, write, and monitor (get, set, and subscribe), to users holding a specific role. A user can be allocated several roles and can select which role they are performing at a given point in time. Once again, the whole mechanism relies on the device-property model to structure its configuration; the rules are organised by device class and class properties. As the device-property model is flexible enough to allow modelling of more than the low-level devices classes (e.g. a FESA class), we can use RBAC to protect any element that can be expressed in terms of class and property. Nevertheless, one has to acknowledge that modelling everything as a class with properties is not always elegant.

RBAC is made of two parts; the client and the server-side. The client-side deals with the authentication of the users and the role selection. Figure 13.1 depicts the login dialog and the RBAC role picker. The server side is in charge of the authorisation, based on the pre-defined rules and the user token provided by the client. As for many elements of the Control System, RBAC relies heavily on the Controls Configuration Database (CCDB) to store roles and rules. As the low-level FECs do not have access to the Database (DB), we extract the rules from the DB into a file, the so-called access map that is read directly by the server-side running on the FECs. RBAC is tightly integrated with Controls Middleware (CMW)-Remote Device Access (RDA) (see section 14.1) and therefore token transport and access control is transparent for the end-users.


RBAC Login Dialog

RBAC Authentication: Please provide your credentials

Username: wsliwins

Password: ●●●●●●●●

RBAC Role Picker

Select roles that you want to use:

☐ Show MCS roles only

- ✓ CMW-ADMIN
- ✓ Critical-Property-Admin
- ✓ DIAMON-Expert
- ✓ DIAMON-FE
- ✓ JMX-EXPERT
- ✓ OASIS-Expert
- ✓ PO-User
- ✓ RBA-ADMIN
- ✓ RBA-COMPUTER-ADMIN
- ✓ RBA-Developer
- ✓ RBA-LOCATION-ADMIN
- ✓ Remote-Reset-User

Figure 13.1: RBAC login dialog and role picker

Originally, RBAC was designed to protect properties of selected devices and is primarily used in frameworks such as FESA and FGC. With time, its scope and usage have grown, and today RBAC also protects high-level services' methods and GUI features. In other words, some Java methods, accessible with RMI, are RBAC-protected to ensure that only users with a given role can call them. This is achieved using Spring interceptor and method annotations. As middle-tier servers have direct access to the DB, the access map is not required, making the implementation lighter. Furthermore, the latest configuration is immediately available. Recently, we started using RBAC to customise GUIs and tailor the available functionality to the user role. In this case, rather than protecting resources, we aim to improve the user experience by hiding unnecessary details and avoid cluttered user interfaces.

The RBAC services must run with an extremely high availability. Therefore, all software runs on two redundant servers and the clients connect randomly to either one. Whenever, one server is unavailable, the clients automatically fall back to the other. While this simple approach to load-balancing works, it's not very flexible and, as for many other services, we will investigate better solutions.

14. Middleware

14.1 CMW-RDA

The Controls Middleware(CMW) is our main communication infrastructure and, in particular the Remote Device Access, CMW-RDA or simply RDA, is the main component used to exchange data between the front-end computers and the high-level controls layers. Other components of the CMW communication infrastructure are the RBAC service, the naming service and several gateways offering protocol conversions, among other things.

The core business of CMW-RDA is to provide a peer-to-peer communication to organise data flow between the distributed components of the Control System. CMW-RDA is the lowest level component that provides the first elements of the device-property model (see chapter 3 for details) and where concepts such as device are exposed to the users. Aside from this, CMW-RDA is purposely completely agnostic of the other CERN-specific concepts such as timing, multiplexing, etc. (see chapter 4). Three operations are supported by CMW-RDA. The first two are the typical command-response operations read and write, which in our environment are called get and set. The third operation comes from the publish/subscribe paradigm and is called subscribe (or monitor). In the case of subscribe, the client indicates its interest to receive updates for a given device-property pair, aka an access point. Once the server has new data, it pushes it to the client (aka update). In addition to the access point, the client can specify a selector (see section 4.3 for details on selectors) for each of the three operations. The selector is mandatory for getting and setting when the device and the property are multiplexed, in order to allow the low-level software to perform the data de-multiplexing. In the case of subscription, the selector is used to filter out some updates when the data's context does not match the selector e.g. data from a different cycle or with a different beam destination. All operation types support synchronous and asynchronous modes. The latter is particularly useful when a large number of access points must be read or written.

CMW-RDA hides the complexity and details of the data exchange between the clients and the device servers. Many typical communication issues are handled by RDA, such as loss of connection, overflows, etc. In addition, a user does not need to know the location of a server (URL) or even which server a given device is on. RDA relies on the CMW directory service to perform the name resolution i.e. from device name to server name and address (URL). The CMW directory service, which is backed up by a database, does more than the simple device-to-server-URL resolution. Indeed, hints can be provided with the query to permit special routing of the clients. Instead of a direct client-server connection, the directory service can route the client to an intermediate node that provides either high-level services or act as a fan-out, shielding the server from the load from multiple clients. The additional services range from a protocol conversion to a complete middle-tier system, providing business logic. Figure 14.1 depicts the typical data exchange that occurs when a user performs a get followed by a set on an access point.

Diving deeper into CMW-RDA's implementation, all low-level networking is encompassed in a layer based on zeroMQ. ZeroMQ is a high-performance asynchronous messaging library on top of TCP/IP. In addition to the support of asynchronous communication between peers, zeroMQ offers great features such as queuing, batching, recovery of connections, and a mostly zero-copy implementation. Its integration in CMW-RDA was eased thanks to the BSD-like API, which will be familiar to anyone knowing the low-level socket API. ZeroMQ is open source with a well-established community and bindings to many languages. It is licensed with LGPL v3 but the authors want to move to Mozilla Public Licence (MPL) v2 to have more freedom. We decided to use zeroMQ for the third version of CMW-RDA following the conclusions of a survey we performed in 2011. ZeroMQ was the most suitable modern lightweight library for networking at the time.

The availability of the CMW-RDA infrastructure is critical as every device access needs it. In order to provide the required service level, the CMW directory service is deployed on two separate computers with client-side load-balancing.

In the previous version (CMW-RDA2), we had an implementation based on the Common Object Request Broker Architecture (CORBA). The main issue with CORBA is that it does not provide asynchronous communication and that caused recurring problems when the receivers (the clients) were too slow to follow the data throughput of the server. Furthermore, CORBA's footprint was bigger, and its implementation was not as efficient e.g. no zero-copy implementation. The end-of-life of the CMW-RDA2 infrastructure is planned for LS2, between 2019 and 2020.

14.1.1 Gateways, Proxies and the Passerelle

CMW proxies were put in place as a way to shield the servers from the varying number of clients. This was necessary at the time, as the FEC's CPUs were not as powerful as they are today, and also due to limitations from the CORBA implementation. With RDA3, proxies are not required, but we still use them during the transition phase in order to allow connection of RDA2 clients to RDA3 servers.

As the LHC experiments use a different middleware, called Data Interchange Protocol (DIP), we also provide gateways to translate RDA data to/from DIP data. This component is a critical link between the LHC Control System and the LHC experiments.

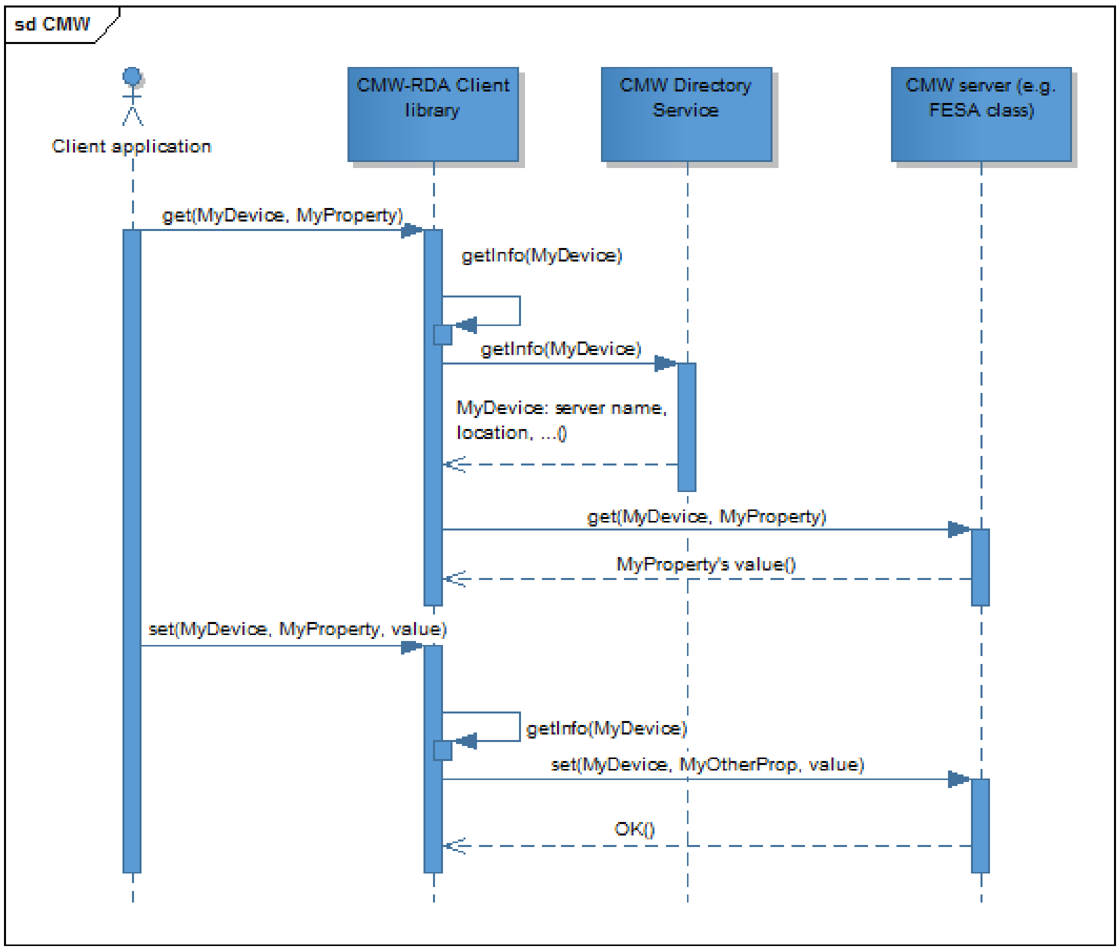

sd CMW
Client application
CMW-RDA Client library
CMW Directory Service
CMW server (e.g. FESA class)
get(MyDevice, MyProperty)
getInfo(MyDevice)
getInfo(MyDevice)
MyDevice: server name, location, ...()
get(MyDevice, MyProperty)
MyProperty's value()
set(MyDevice, MyProperty, value)
getInfo(MyDevice)
set(MyDevice, MyOtherProp, value)
OK()

Figure 14.1: Data exchange during CMW-RDA get/set

Finally, we also provide a .NET plug-in to allow clients running on MS Windows to use CMW-RDA directly. This library, called the Passerelle, lets operators and accelerator experts acquire data directly from tools such as MS Excel and Mathematica. However, as the need for such a component is gradually disappearing, the plan is to phase out the Passerelle.

14.2 JMS

In the early 00s, as RDA2 had some limitations, mainly with its scalability, we needed a middleware solution which scales better. We started using Java Messaging Service (JMS) as a way to serve data from Java services, something that was not easily possible with RDA2 as the server-side implementation was incomplete.

As indicated in its name, JMS focuses on Java-to-Java communications, but it is not a Java-exclusive solution, as bindings for many languages are available. In addition, JMS has an architecture that tackles the scalability issue natively (one producer, many consumers) and offers guaranteed delivery. With JMS, the server sends the data (a message) to a broker and the broker forwards it as many times as needed to the clients. Data is posted to either queues or topics, and clients subscribe, using the same queues and/or topics. We mainly rely on topics, as a topic distributes the messages to each and every client, while the queue sends one message to just one client before removing it from the queue. For many years, JMS has been the standard BE-CO approach whenever communication between two Java servers, or a Java server and Java GUIs, was necessary. As JMS is geared towards the publish-subscribe paradigm, the typical setup was RMI for command-response and JMS for data publication from the server. Looking back at the examples given chapter 2, the data flows from the low-level (FECs) are post-processed by middle-tier servers before finally being sent onwards to be displayed in a GUI. Most of the time, the amount of data is greater after the post-processing, as the data is either augmented with information available to the middle-tier servers (e.g. from databases) or the representation is at a higher-level of abstraction, more suitable for GUIs and high-level applications (e.g. no encoding of several pieces of info into a single integer).

The main issue with the client-server architecture used by CMW-RDA is that it does not scale horizontally. It relies on the fact that the publishing point (i.e. the FEC in most of the case) has enough resources to send the data as many times as there are clients. If the number of clients varies a lot, this can create a load problem that is not trivial to solve. Estimating whether a given system will be able to fulfil its real-time constraints and cope with the client load is extremely difficult if the number of clients, hence the amount of data to send, can vary by an order of magnitude. The JMS solution brings the intermediate broker or brokers into the equation and the broker(s) handle the additional load. Figure 14.2 depicts the current JMS infrastructure.

Of course, being shielded from the client load variation has a cost and, in this case, JMS introduces an additional delay in the communication. Most of the time, this is not a problem but, for systems such as OASIS (see chapter 23 for details) where the control room operators examine several sources of information at the same time, it is important to keep the updates' time difference within a quarter of second, and this is not always possible with JMS.

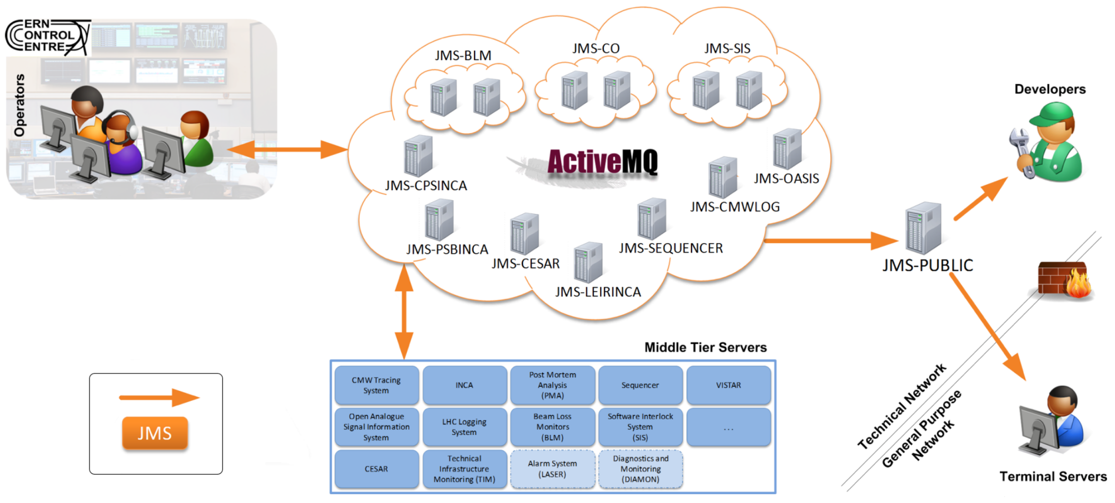


Figure 14.2: JMS infrastructure and its main users

Also, while it is very convenient to have a strong decoupling between the client and the server, one has to be aware of the advantages and drawbacks. Its flexibility means that you can easily start a client before the server is even deployed and it is very easy to add JMS topics for clients to listen to. Compared to CMW-RDA, where everything is tightly integrated and every device must be declared in the DB, JMS is more flexible and can be more suitable for some applications. On the other hand, if your client does not receive data, it may be because the topic name is misspelt despite being successfully connected to a topic. Development and diagnostics can be harder as the client and server parts are further decoupled. Nevertheless, we can run the broker embedded in the server process during the development and test phases and Java Management Extensions (JMX) metrics, published by a web server, help to alleviate the problems.

A side effect of the JMS horizontal scalability, with data being sent from broker to broker, is that it allows us to have a clean solution for forwarding data from the Technical Network (TN) to the General Purpose Network (GPN). The solution consists of having only one broker (JMS-PUBLIC) with GPN connectivity. The other brokers on the TN are not accessible from the GPN broker and forward the required data to JMS-PUBLIC. In this way, the TN services remains isolated from the connection with the GPN.

The first brokers were introduced in the early 00s and the first JMS deployments were based on SonicMQ. Later, we moved to Free Open Source Software and used the Apache software foundation’s ActiveMQ. This broker offers all the features we need and more besides, such as SSL, multiple endpoints, and policies. The latter is a feature not available in RDA that allows better control of the messages’ characteristics such as their size and rate, as well as how many clients, etc. ActiveMQ also supports many protocols; Openwire and Stomp being those used at CERN.

In 2019, we have approximately 25 JMS brokers with a broad range of usage with as many as 1000s of messages per second and up to 2TByte per day. In the majority of cases, we have one broker per service (e.g. one broker for the PS Booster InCA server, one for the LEIR server, etc.). While redundancy and clustering would be possible, this kind of setup is more complex and introduces more delay. With the stability of the brokers being excellent, there is no reason to implement this unless a specific application requires it, such as the LHC Beam Loss Monitor (BLM) concentrator or the LHC Software Interlock System (SIS).

Nowadays, alternative solutions exist and a study is under way to evaluate whether we can replace JMS by RDA3. In this case, we are aware that we will lose the advantages brought by the broker-based architecture, but it would allow us to rationalise the technologies used in Controls stack.

14.3 JAPC

The Java API for Parameter Control (Java API for Parameter Control (JAPC)) is an in-house, client-side Java library that was developed around 2003. The primary purpose of JAPC is to offer an abstraction layer on top of the different middleware (see sections 14.1 and 14.2). The two main JAPC extensions are for CMW-RDA (japc-ext-rda) and JMS (japc-ext-remote). Nevertheless, it is possible to wrap any libraries exposing an API with get and set methods and call-backs, into a JAPC extension, as depicted in figure 14.3. This concept has been successfully used to wrap the complex API of the timing library (TGM).

Thanks to this extension, the end-user has only to learn the JAPC API.

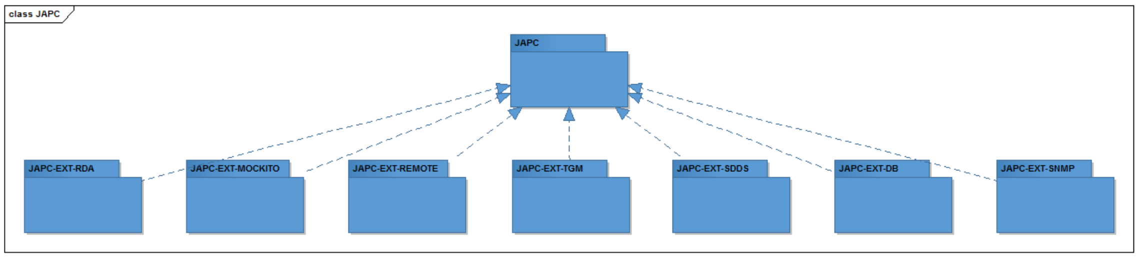


Figure 14.3: JAPC's main extensions

JAPC is not a device-property API but instead it is designed around the concept of a parameter. Typically, a JAPC parameter is made by pairing a CMW device with a property. For example, for the device MY.DEVICE and its property Prop1, the corresponding JAPC parameter name would be MY.DEVICE/Prop1. Performing a get operation on this parameter returns an object implementing the ParameterValue interface, which gives access to the different value-items contained in the property (see chapter 3 for details on the device-property model). It is worth noting that, unfortunately, the property's value-items are referred to as fields in JAPC and in the higher layers of the Control System. As often the whole property is not needed, JAPC offers the possibility to create parameters for a value-item (field) directly. To come back to our example, to read the value-item value1, the full parameter name would be MY.DEVICE/Prop1#value1. One has to realise that, when the underlying middleware is CMW-RDA, retrieving a single field in the property will not save any network bandwidth as the whole property is always returned; in CMW-RDA, the only access point is at the level of the device-property pair. Furthermore, one must be aware that, when setting new values, the complete property might have to be set, which is recommended in order to maintain the atomicity of the property. Obviously, if the underlying device is multiplexed, accessing it will require a cycle selector. In order to support multiplexing, JAPC provides a selector class to specify the context in which an operation must be executed. Figure 14.4 depicts the main JAPC interfaces with their relationships.

JAPC also extends the features offered by the middleware. Thanks to the ability to add descriptors to the parameters, JAPC supports value types at a higher level of abstraction. For example, if a FESA class describes an enumeration, this description can be provided to JAPC through a descriptor. This allows the client to work with the enumeration and not only the integer that is transported by CMW-RDA.

JAPC is also not limited to the basic types that CMW-RDA supports and can transport serialised Java objects. This feature is very useful when a Java server exposes a JAPC API but requires more complex data structures. However, for this to work, the Java class must be available on the client side in order to be able to reconstruct (deserialise) the object.

JAPC relies on the CMW directory service to select the appropriate extension or middleware to be used. Nevertheless, for more complex cases, JAPC supports custom resolvers that allow the routing to be modified. For example, allowing calls to a CMW device to be redirected to a Java server (e.g. the InCA server) in order to perform additional actions (e.g. trim history, setting computation, etc.) before sending them to the CMW server.

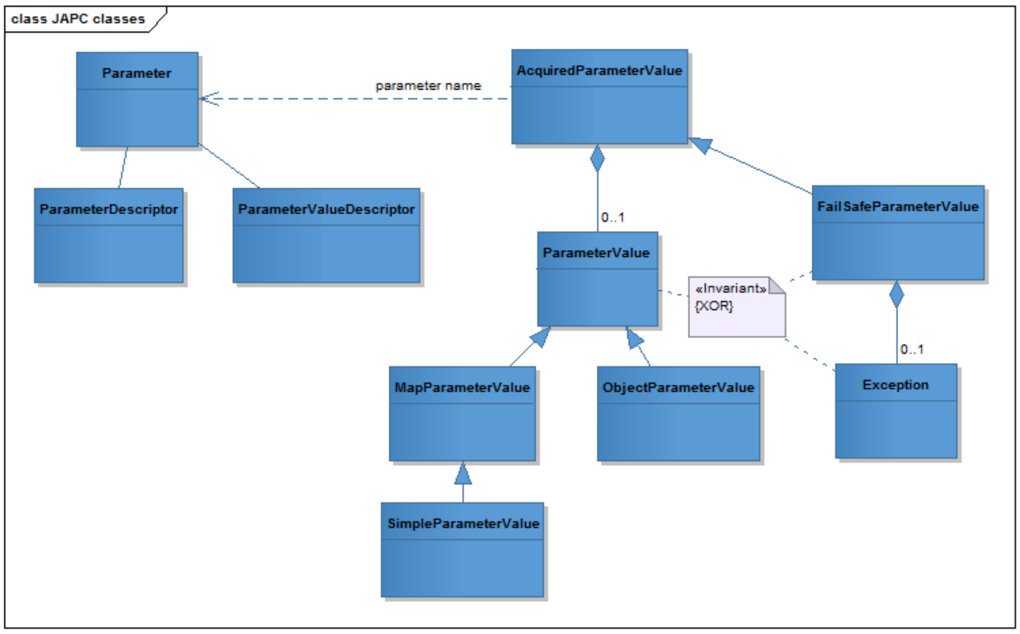


Figure 14.4: JAPC's main interfaces and their relationships

JAPC also supports parameter grouping. Multiple parameters can be assigned to a group in order to perform batch operations. The implementation for get/set operations is relatively trivial. However, it gets much more complicated for the subscriptions, as the updates of the different JAPC parameters need to be grouped accordingly. As explained in section 4.1, our accelerator chain produces beams, therefore, it is natural to group the updates per beam occurrence. At an accelerator level, a beam is a cycle and the beginning of every cycle is timestamped (cycle stamp or cycle timestamp). JAPC uses the cycle stamp to group the updates and extract meta-information from the incoming data, which is then stored in the header of the parameter value. Similar processing is also done for other types of meta-data, such as the min/max and units, but this is based on value-item naming conventions.

To ease the decoupling of the application during testing, we developed a JAPC extension that can simulate device access. One can develop a scenario, including updates, to be played during the tests, as if the real devices were responding. Also, following the new trend of using reactive streams, we recently developed a layer on top of JAPC (japc-stream) that allows you to subscribe, process and republish JAPC parameters with a stream API. This solution is quite elegant and the fact that it is based on JAPC means that all of the data sources using the various middleware are available.

Finally, the narrow interface exposed by JAPC is perfect for generic, data-driven applications. Nevertheless, for specific applications, it is unfortunate not to be able to reuse the information available in the FESA class design, for example, to profit from compile-time error checks. For these cases, the FESA tools provide the possibility to generate a set of Java classes from the FESA class design (the JAPC beans). This is a valid alternative to JAPC descriptors, allowing the developer to profit from Java features and work directly with real Java classes. However, this means it is necessary to maintain a FESA class specific library (JAR file) containing the generated classes.

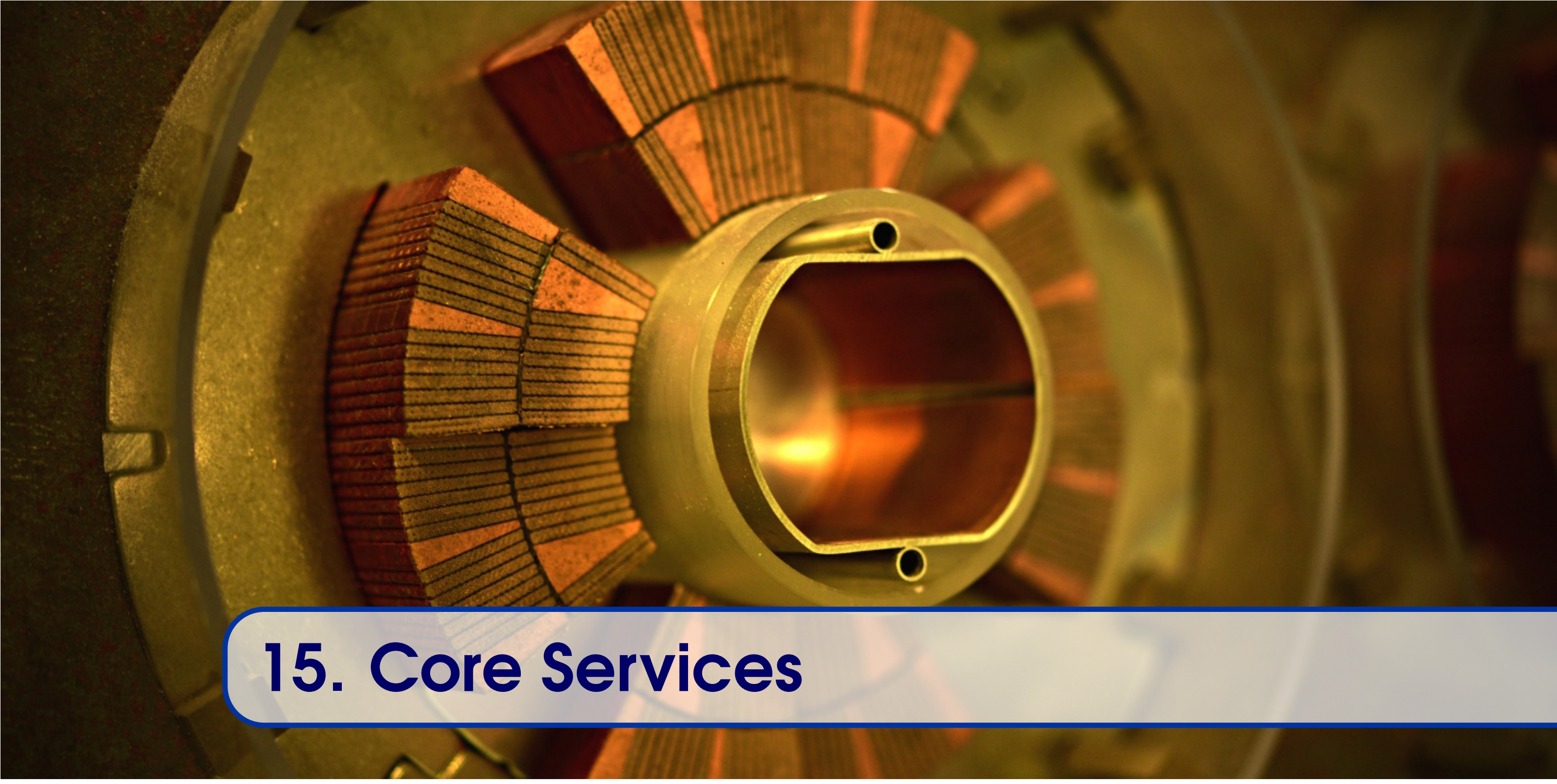
14.3.1 JAPC Monitoring

There are many situations where one needs to monitor several JAPC parameters and post-process the results. The JAPC monitoring (JMON) layer was developed to cover this need and improve the diagnostics.

Based on a configuration file, the JMON library subscribes to device-properties in the Control System and sends the updates to JAPC monitoring modules. The users of the library provide custom modules in order to synchronise and process the data. One of the advantages of using JMON is that the module code is isolated from the actual data source, thanks to the possibility to use symbolic names in the configuration file. In addition, JMON comes with a monitoring GUI to help with fault-detection and subscription management (start, stop and restart).

V

High-Level Software



15. Core Services

The components described in this chapter constitute the core high-level services that allow the control and monitoring of the low-level components described in previous chapters. They are the foundations for most of the high-level applications.

15.1 InCA/LSA

Depending on the accelerator, slightly different solutions exist in order to take into account the differences in accelerator operation. For the LHC and the SPS, the core high-level Control System is centred on settings management via the LHC Software Architecture (LSA) [39]. For the lower-energy accelerators, LSA is integrated into the Injector Controls Architecture (InCA) [19, 38]. In addition, InCA provides services for accelerator monitoring and configuration of generic applications (see section 17.2).

15.1.1 Setting Management

As described in chapter 4, CERN's accelerators run in parallel and produce different beams for different experiments at the same time. Furthermore, the number of beam types that can be produced during an LHC run (typically 5 years) is huge, and the settings for all of those beams cannot be stored in the limited memory of the FECs. During the lifetime of a beam type, many modifications and optimisations, called trims, will be performed. It is important to keep a record of the trims in order to understand what has been done and, if necessary, to be able to revert to a previous situation. For these reasons, the Control System must provide a solid settings management solution.

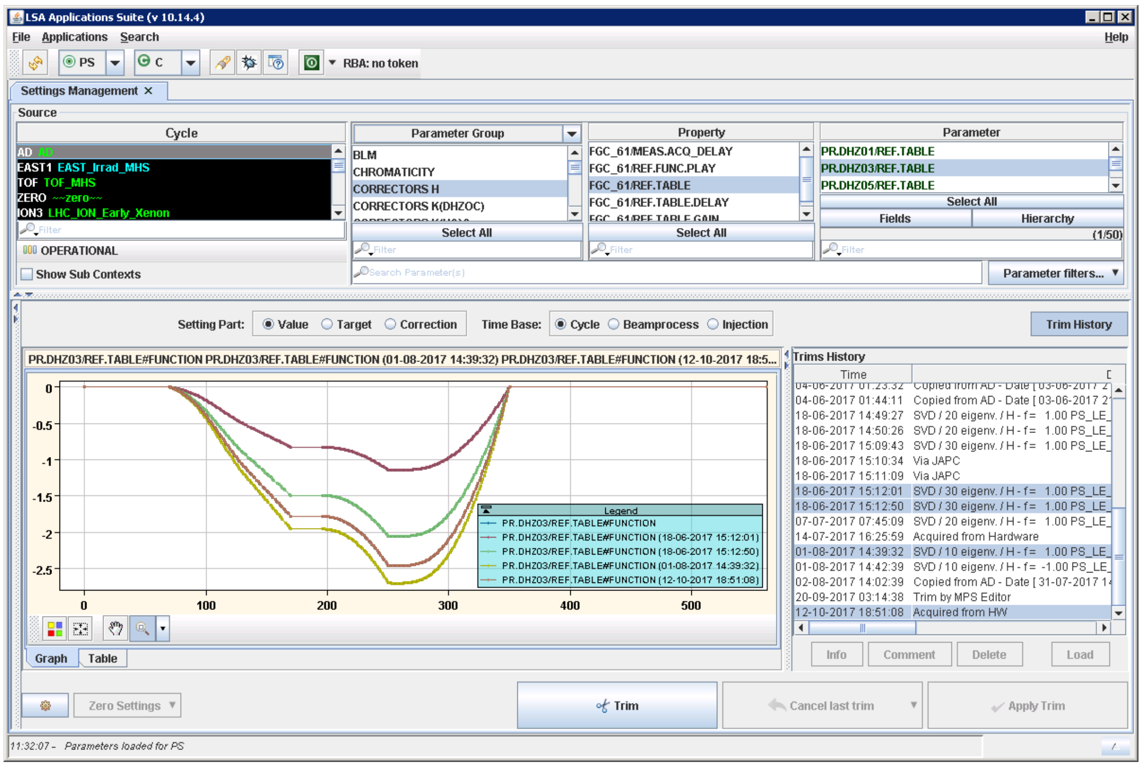
The settings are the setpoint values for the hardware that is used to control and monitor the beams. The majority of the operational devices (see chapter 3) have their settings managed by LSA. In LSA, a parameter is equivalent to the field (or value-item) in a property of a device. For a parameter used in accelerator control, the setting management stores not only the current values for all cycles of the accelerator, but also all of the previous values in the trim history. Thanks to the trim history, one can know the exact state of the accelerator at any moment in the past. Furthermore, the system allows the user to revert to any point in time by sending the historical settings to the low-level Control System. Figure 15.1 shows an example of the setting management application in the LSA Application Suite. To simplify the work of the operators, there is a possibility to indicate that the current set of settings give certain results and therefore to mark those settings as a reference. For each cycle, one has the possibility to mark a single value as a reference for a given parameter. It is also possible to label the current set of settings for a given cycle with a name, and create what is called an archive. The archive and references can be used to highlight differences between the current setup and to restore the accelerator or a sub-system to a previously known state.

So far, we have described what is referred to as low-level setting management. More can be done by offering an additional layer of abstraction thanks to the concept of a high-level parameter. High-level parameters are typically more oriented towards the physics of the accelerator, rather than the implementation details, as exposed by the front-end computers. LSA takes care of translating values of these high-level parameters into the corresponding low-level parameters, such as currents to be produced by power converters, through parameter hierarchies. This allows the direct modification of accelerator properties, such as the tune or chromaticity, instead of adjusting many low-level parameters. Another example of a high-level parameter is to control the beam position at a given location in an accelerator, and the system takes care of computing the deflecting angles and the current modifications required to achieve them.

Figure 15.2 depicts a parameter hierarchy linking the current of the low-level power converter with the horizontal chromaticity of the SPS accelerator. For each relationship, one can define an algorithm, called a makerule, to transition from one level to the next. Often, these algorithms need additional information related to the optics of the accelerator. The different optics are available in the LSA database and are calculated by the MAD X application using data from the Layout database (see chapter 21).

Defining a complete high-level model of an accelerator allows the operators to work with physics parameters without having to be aware of low-level details. This also allows them to describe the cycles they would like to play, and generate an initial set of settings corresponding to the cycle described. However, this approach is imperfect as the model is never completely accurate and once the cycle is sent to the low-level layer, some trims are always required. For some of the oldest accelerators, the discrepancy between the best available model and the real behaviour of the accelerator is so wide that the resulting initial set of settings calculated from the model is not very useful. Instead, for these accelerators, the operators copy settings from one cycle to another, thus profiting from previously optimised values.

A trim process is comprised of several steps. First, the new high-level parameter value is received and validated. Then, the parameter hierarchy in which the trimmed parameter is involved is retrieved. By invoking the makerules, the low-level parameter values are recalculated; this is the actual trim operation. Once all of the new values are available,



Cycle	
AD	AD
EAST1	EAST_Irrad_MHS
TOF	TOF_MHS
ZERO	~~zero~~
ION3	LHC_ION_Early_Xenon

OPERATIONAL

Show Sub Contexts

Parameter Group: BLM, CHROMATICITY, CORRECTORS H, CORRECTORS K(DHZOC)

Property: FGC_61/MEAS.ACQ_DELAY, FGC_61/REF.FUNC.PLAY, FGC_61/REF.TABLE, FGC_61/REF.TABLE.DELAY, FGC_61/REF.TABLE.GAIN

Parameter: PR.DHZ01/REF.TABLE, PR.DHZ03/REF.TABLE, PR.DHZ05/REF.TABLE (1/50)

Setting Part: [✓ Value] Target Correction   Time Base: [✓ Cycle] Beamprocess Injection

PR.DHZ03/REF.TABLE#FUNCTION PR.DHZ03/REF.TABLE#FUNCTION (01-08-2017 14:39:32) PR.DHZ03/REF.TABLE#FUNCTION (12-10-2017 18:5...

PR.DHZ03/REF.TABLE#FUNCTION
PR.DHZ03/REF.TABLE#FUNCTION (18-06-2017 15:12:01)
PR.DHZ03/REF.TABLE#FUNCTION (18-06-2017 15:12:50)
PR.DHZ03/REF.TABLE#FUNCTION (01-08-2017 14:39:32)
PR.DHZ03/REF.TABLE#FUNCTION (12-10-2017 18:51:08)

Trims History

Time	D
04-06-2017 01:23:32	Copied from AD - Date [03-06-2017 2
04-06-2017 01:44:11	Copied from AD - Date [03-06-2017 2
18-06-2017 14:49:27	SVD / 20 eigenv. / H - f= 1.00 PS_LE_
18-06-2017 14:50:26	SVD / 20 eigenv. / H - f= 1.00 PS_LE_
18-06-2017 15:09:43	SVD / 30 eigenv. / H - f= 1.00 PS_LE_
18-06-2017 15:10:34	Via JAPC
18-06-2017 15:11:09	Via JAPC
18-06-2017 15:12:01	SVD / 30 eigenv. / H - f= 1.00 PS_LE_
18-06-2017 15:12:50	SVD / 30 eigenv. / H - f= 1.00 PS_LE_
07-07-2017 07:45:09	SVD / 20 eigenv. / H - f= 1.00 PS_LE_
14-07-2017 16:25:59	Acquired from Hardware
01-08-2017 14:39:32	SVD / 10 eigenv. / H - f= 1.00 PS_LE_
01-08-2017 14:42:39	SVD / 10 eigenv. / H - f= -1.00 PS_LE_
02-08-2017 14:02:39	Copied from AD - Date [31-07-2017 1
20-09-2017 03:14:38	Trim by MPS Editor
12-10-2017 18:51:08	Acquired from HW

11:32:07 - Parameters loaded for PS

Figure 15.1: Settings Management application showing the trim history of a power converter in the PS

the system sends the new low-level parameter values to the Front-End Computers; this operation is called a drive. During the drive, the front-end computers receive the new low-level settings and perform additional checks before accepting the values. The first part, the trim, is executed in a single node of the Control System and therefore, if a calculation fails or yields out-of-bound values, the whole transaction can easily be rolled back. However, apart from a few exceptions, there is no distributed transaction support and once the values are driven, if an individual FEC refuses a value or crashes, it is not currently possible to roll back the whole process. The policy to deal with this type of situation is accelerator-dependant and can be to either continue and report the error, or to roll back the trim operation. Neither of these policies are perfect as it is still possible to end up with some FECs having settings that differ to those stored in the settings management system.

Contrary to the FECs, which can only have a limited number of timing users, the setting management can handle as many cycles as needed. There is a mapping between the cycle, which only exists at high-level, and the timing user in the low-level front-end computer (see section 4.3). A cycle that is not associated or mapped to a timing user is called a non-resident cycle. It is still possible to work with a non-resident cycle, as the low-level parameters can still be calculated. However, once the calculation is finished, there is no drive, and as such the validation of the new value by the front-end computers is not done; making the operation more theoretical and reserved mainly for cycle preparation.

A setting context is a container for settings for a single beam in a single accelerator. The

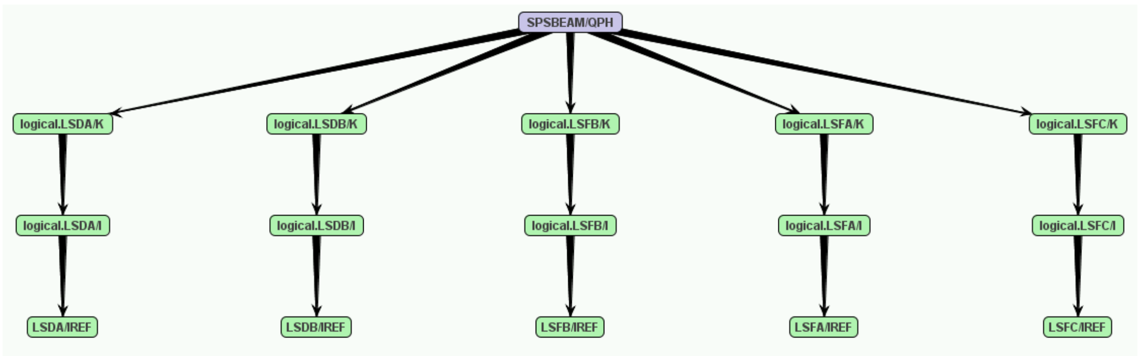


Figure 15.2: Parameter hierarchy for the SPS horizontal chromaticity control

type of setting context depends on the nature of the accelerator (cycling machine, collider, linac, etc.) and, for the majority of CERN accelerators, a setting context corresponds to a cycle. However, for machines such as the LHC, where the cycle length is not predetermined, as it depends on the machine performance, the setting context does not correspond to the whole cycle. Instead, the cycles are assembled progressively from a set of beam processes. In this case, LSA uses beam process as the setting context. In addition, even for accelerators that have cycles as setting contexts, the beam process concept can still be used to build cycles.

From an operational point of view, the parts of the cycles where there is no beam in the machine is less important. In these cases, we want to reach the next level as fast as possible, taking into account the hardware constraints, e.g. the ramping rate of the power converters. These parts of the cycle are known as "BeamOut" and are computed by LSA using link rules. Figure 15.3 depicts an SPS Super Cycle decomposed into several beam processes; the BeamOuts are shaded.

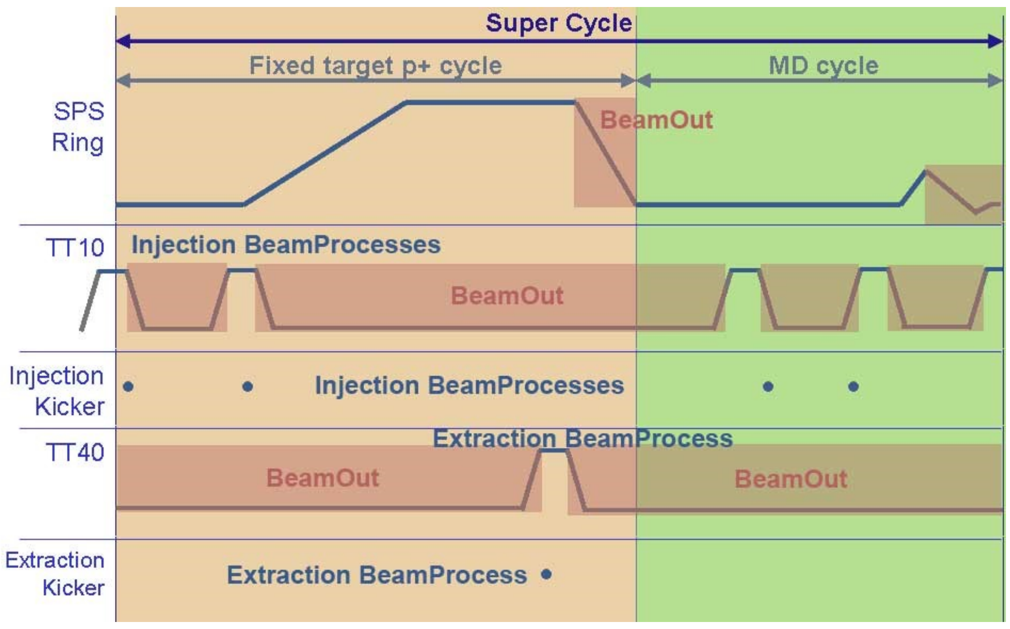


Figure 15.3: Beam processes in a typical SPS supercycle

Since 2017, all of the accelerators have their settings managed by LSA. Starting in 2003, the development of LSA initially focused on the SPS and the LHC, with the SPS used to validate the system under development. From 2008, it was decided to homogenise settings management and to extend the functionality of LSA to cover other accelerators such as the PS, the AD, etc. The current implementation is based on a Java server and a relational database as depicted in Figure 15.4.

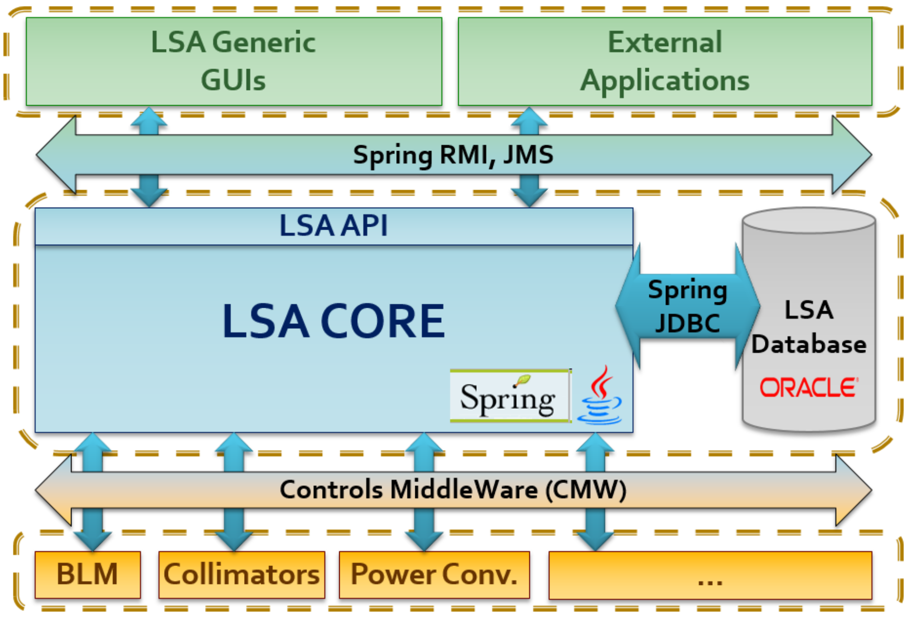


Figure 15.4: LSA high-level architecture

15.1.2 Acquisition and Monitoring

For the smaller accelerators, operators have stronger requirements for the Control System to provide a constant flow of acquired values. In addition to acquiring the values, post-processing must be done in order to indicate the status of the machine or a subset of elements. This constitutes the second main aspect of InCA and is commonly referred to as the Acquisition Core.

Front-end computers are much less powerful than the available servers and consoles, even though since 2012, this has been improved with access to multi-core SBCs with 1GB+ of RAM. It often occurred that too many clients were requesting the same data for the same timing user, from the same set of devices, at the same time. This, coupled with scalability issues in CMW-RDA2, led to an unstable operational environment with many applications cycling between disconnection and reconnection. Taking into account that the PS complex's operators wanted to monitor many accelerators' low-level parameters and display them in tables called WorkingSets (see section 17.2.1), it was decided to build the Acquisition Core (AcqCore).

The AcqCore subscribes once, and only once, to a given device-property with the ALL selector, i.e. without timing selector filtering, instead of every client subscribing to their parameters for their specific timing user. Furthermore, to prevent the FECs from being disturbed by constant requests to subscribe and unsubscribe, when a client disappears, the subscription is maintained for a given duration, as it is likely that the same parameters will be required by another client. This means that the FECs have a more stable load as the maximum network load is now defined as the ability to send all of the device-properties once every timing user. Therefore, we are now able to verify that a FEC scales correctly up to its maximum load.

The AcqCore is also responsible for enhancing the acquired data. This can be done in two ways, by status computation or through virtual acquisition parameters. Status computation is when the AcqCore compares the acquired value, which can be either a setting or an acquisition parameter, with either the reference value or the corresponding setting value respectively. This comparison, taking into account associated tolerances, gives an indication of the accelerator's behaviour; are the quadrupoles producing the requested current? Which elements in this transfer line are not set to their reference value? The calculated status information is sent to the User Interfaces (UIs) and represented in the graphical components as a background colour. Figure 15.5 depicts part of the status algorithm as defined in the initial specifications of InCA.

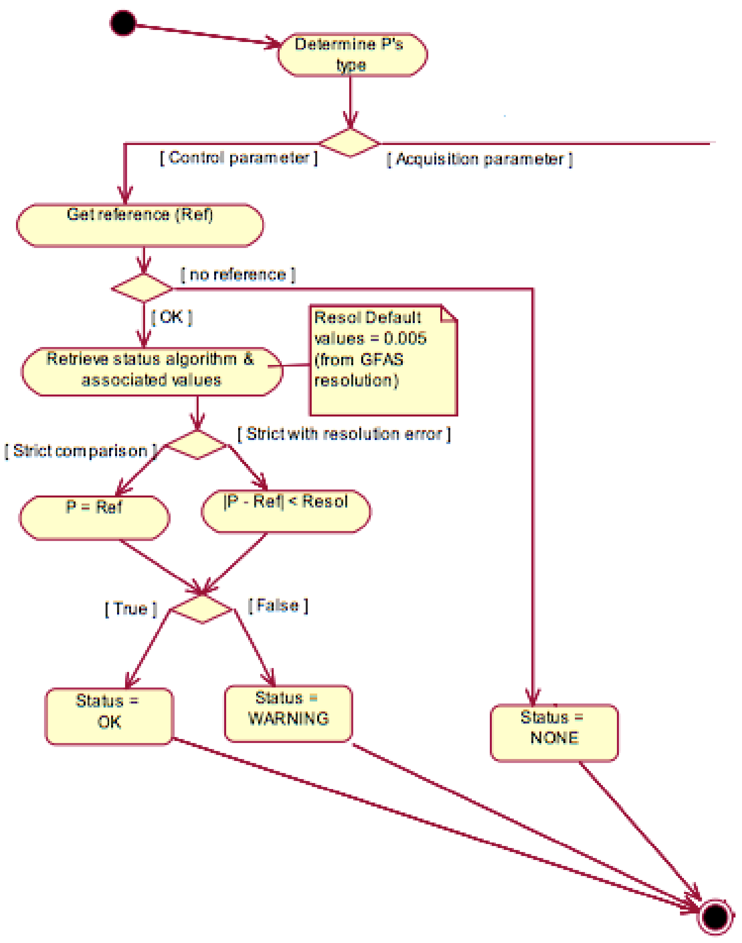


Figure 15.5: Status calculation for setting parameters

The virtual acquisition parameters are the acquisition counterpart of the high-level control parameters used in parameter hierarchies but working bottom-up instead of top-down. As for the control parameters, algorithms are attached to the transitions. In this case, they are called calculation rules instead of makerules. For example, in the ISOLDE machine, four magnets had to be controlled so that they appeared to behave like a single quadrupole. Settings are calculated as LSA makerules and a calculation rule is used to compute the average current from the individual power converter acquisitions.

The AcqCore's implementation has to be scalable, as we typically process several thousand values every cycle. Acquisition values arrive throughout the cycle and some values requiring post-processing may arrive during the next cycle. To maintain the user-experience, the system is triggered every 250ms to perform the evaluations of the parameters already available and distribute the result to the clients. After a grace period, well into the following cycle, exceptions are generated for parameters that were not updated for the last cycle.

As for most of our high-level services, the AcqCore is driven by a database and one of the challenges is to perform live configuration updates without disturbing the service.

15.2 CESAR

In the experimental areas such as the SPS north area and the PS east areas, we have a number of short-term experiments taking place in succession. They are short-term when compared with the LHC experiments that were installed with the accelerator itself and will remain in place for its entire life (25 years).

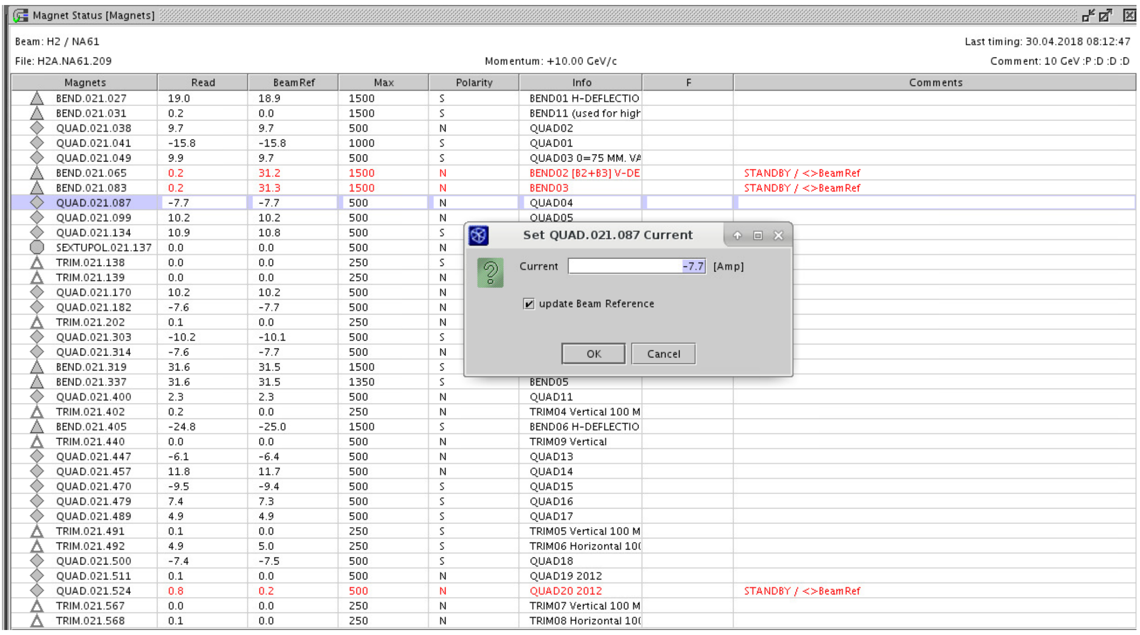
An accelerator operator is hired to expertly run the accelerator and needs a deep understanding of how the machine and its Control System work. A physicist typically comes to CERN for a short period of time in order to conduct an experiment and only has to be able to perform a limited set of control actions on his beamline. Therefore, the physicists need a simplified Control System. Many controls concepts such as the machine multiplexing can be safely ignored when operating the experiments, and so the user interface and feature set can be greatly simplified.

For these reasons, in 2000, the CERN Experimental Areas Software Renovation (CESAR) project was launched in order to provide modern high-level software for the experimental area controls. It is worth noting that the low-level controls are based on the classic stack, as described in this document, since the implementation details of the low-level layers can be hidden from the end-users.

In order to make things even simpler, CESAR implements specific virtual devices hiding the complexity of the low-level devices. For example, if a collimator is realised with two motors, CESAR exposes a virtual device representing the collimator, concealing the motors from the users. This gives a better integrated and more user-friendly system, which is easier to configure, but the drawback is that custom code is necessary whenever a new type of equipment is installed in the beam line.

Less experienced users are restricted to operating equipment within very specific zones. A lightweight access control, relying on the CESAR's knowledge of the beam line structure, defines the zones and the range of equipment that the user can work with.

In addition, CESAR provides features that are specific to experimental areas such as scans and beam files. Scans are typically performed in experimental areas when the expert physicists set up the beam for an experiment and try to optimise a set of parameters by scanning the range of values of other ones. For example, as shown in figure 15.6, one can optimise the beam transmission by changing the current in the quadrupole magnets. We have custom scans for each type of equipment, such as magnets, collimators, etc. The scan implementation handles all logic to automate the scan and the only thing required from the user is to describe what he wants to do, through its API. Internally, the scan package validates that the device is ready, sets the new value, waits for the device to reach its steady state, measures the corresponding results, and verifies that the results are correct. The output, the raw data, is displayed on a graph.


Magnet Status [Magnets]

Beam: H2 / NA61
File: H2A.NA61.209
Momentum: +10.00 GeV/c
Last timing: 30.04.2018 08:12:47
Comment: 10 GeV :P :D :D :D

Magnets	Read	BeamRef	Max	Polarity	Info	F	Comments
△ BEND.021.027	19.0	18.9	1500	S	BEND01 H-DEFLECTIO		
△ BEND.021.031	0.2	0.0	1500	S	BEND11 (used for high		
◇ QUAD.021.038	9.7	9.7	500	N	QUAD02		
◇ QUAD.021.041	-15.8	-15.8	1000	S	QUAD01		
◇ QUAD.021.049	9.9	9.7	500	S	QUAD03 0=75 MM. VA		
△ BEND.021.065	0.2	31.2	1500	N	BEND02 [B2+B3] V-DE		STANDBY / <>BeamRef
△ BEND.021.083	0.2	31.3	1500	N	BEND03		STANDBY / <>BeamRef
◇ QUAD.021.087	-7.7	-7.7	500	N	QUAD04		
◇ QUAD.021.099	10.2	10.2	500	N	QUAD05		
◇ QUAD.021.134	10.9	10.8	500	S			
⬡ SEXTUPOL.021.137	0.0	0.0	500	N			
△ TRIM.021.138	0.0	0.0	250	S			
△ TRIM.021.139	0.0	0.0	250	N			
◇ QUAD.021.170	10.2	10.2	500	N			
◇ QUAD.021.182	-7.6	-7.7	500	N			
△ TRIM.021.202	0.1	0.0	250	N			
◇ QUAD.021.303	-10.2	-10.1	500	S			
◇ QUAD.021.314	-7.6	-7.7	500	N			
△ BEND.021.319	31.6	31.5	1500	S			
△ BEND.021.337	31.6	31.5	1350	S	BEND05		
◇ QUAD.021.400	2.3	2.3	500	N	QUAD11		
△ TRIM.021.402	0.2	0.0	250	N	TRIM04 Vertical 100 M		
△ BEND.021.405	-24.8	-25.0	1500	S	BEND06 H-DEFLECTIO		
△ TRIM.021.440	0.0	0.0	500	N	TRIM09 Vertical		
◇ QUAD.021.447	-6.1	-6.4	500	N	QUAD13		
◇ QUAD.021.457	11.8	11.7	500	N	QUAD14		
◇ QUAD.021.470	-9.5	-9.4	500	S	QUAD15		
◇ QUAD.021.479	7.4	7.3	500	S	QUAD16		
◇ QUAD.021.489	4.9	4.9	500	S	QUAD17		
△ TRIM.021.491	0.1	0.0	250	S	TRIM05 Vertical 100 M		
△ TRIM.021.492	4.9	5.0	250	S	TRIM06 Horizontal 10(		
◇ QUAD.021.500	-7.4	-7.5	500	S	QUAD18		
◇ QUAD.021.511	0.1	0.0	500	N	QUAD19 2012		
◇ QUAD.021.524	0.8	0.2	500	N	QUAD20 2012		STANDBY / <>BeamRef
△ TRIM.021.567	0.0	0.0	250	N	TRIM07 Vertical 100 M		
△ TRIM.021.568	0.1	0.0	250	N	TRIM08 Horizontal 10(		

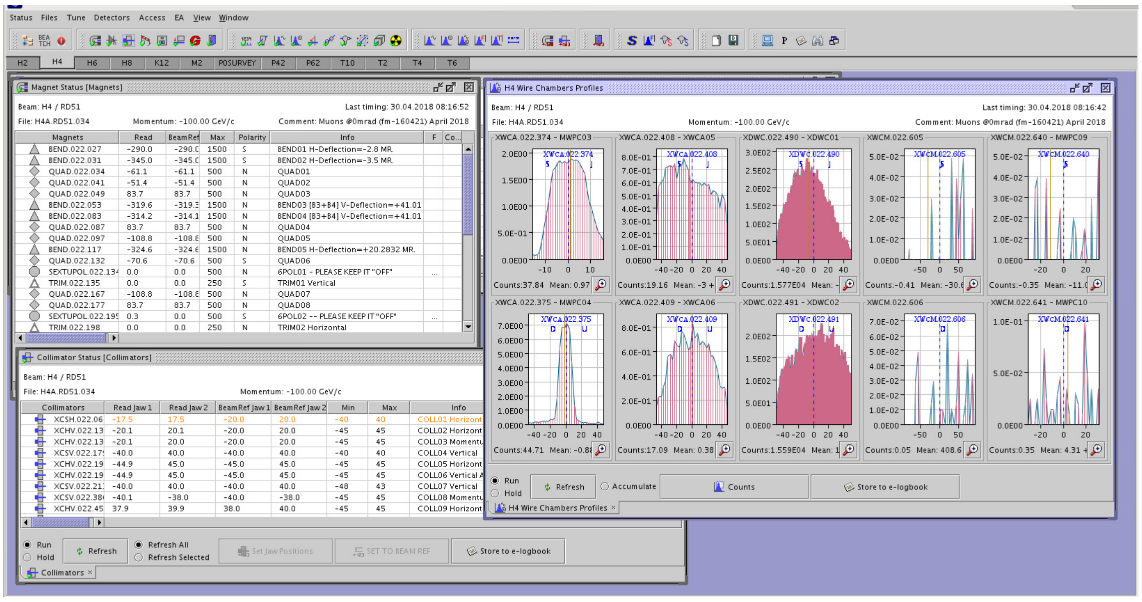
Set QUAD.021.087 Current
Current -7.7 [Amp]
☑ update Beam Reference

Figure 15.6: Optimising the beam transmission by changing the current in a quadrupole magnet

A beam file contains all of the settings for a whole beam line. They can be seen as a bag of settings without history. Furthermore, the user can create archives of beam files and reload them, or part of them, when necessary using the mechanism provided by CESAR. The mechanism handles the different steps such as turning off the beam, restoring the settings, etc. The user can also modify the beam files offline with an editor or Excel.

CESAR is based on the technologies that were available and recommended by BE-CO at the time of its development [7, 28]. At the core of the system is a client-server architecture written in Java. The noteworthy difference between CESAR and LSA is that the former is not database-driven. Indeed, the implementation of the logic is based on custom Java code and heavily uses the FESA bean generation, providing type-safe property access; something not easily done with a database-driven system. The GUI, depicted in figure 15.7, relies on an old version of Netbeans (i.e. Swing). As for the other client-server systems developed in the early 2000s, the communication is based on RMI for the command/response and on serialised Java objects sent via JMS for the server-client communication.

Some consolidations of the system are foreseen, such as the replacement of the GUI and a review of settings management, which could be merged into LSA. Nevertheless, unifying the settings management will not remove the need for the simplified controls environment provided by CESAR. The possibility to base access control on the RBAC service (see section 13.2 for details), instead of the current custom solution, needs to be investigated.



Magnet Status [Magnets]

Beam: H4 / RD51 Last timing: 30.04.2018 08:16:52
File: H4A.RD51.034 Momentum: -100.00 GeV/c Comment: Muons @0mrad (fm-160421) April 2018

Magnets	Read	BeamRef	Max	Polarity	Info	F	Co...
BEND.022.027	-290.0	-290.0	1500	S	BEND01 H-Deflection=-2.8 MR.		
BEND.022.031	-345.0	-345.0	1500	S	BEND02 H-Deflection=-3.5 MR.		
QUAD.022.034	-61.1	-61.1	500	N	QUAD01		
QUAD.022.041	-51.4	-51.4	500	N	QUAD02		
QUAD.022.049	83.7	83.7	500	N	QUAD03		
BEND.022.053	-319.6	-319.3	1500	N	BEND03 [B3+B4] V-Deflection=+41.01		
BEND.022.083	-314.2	-314.1	1500	N	BEND04 [B3+B4] V-Deflection=+41.01		
QUAD.022.087	83.7	83.7	500	N	QUAD04		
QUAD.022.097	-108.8	-108.8	500	N	QUAD05		
BEND.022.117	-324.6	-324.6	1500	N	BEND05 H-Deflection=+20.2832 MR.		
QUAD.022.132	-70.6	-70.6	500	S	QUAD06		
SEXTUPOL.022.134	0.0	0.0	500	N	6POL01 - PLEASE KEEP IT "OFF"	...	
TRIM.022.135	0.0	0.0	250	S	TRIM01 Vertical		
QUAD.022.167	-108.8	-108.8	500	N	QUAD07		
QUAD.022.177	83.7	83.7	500	N	QUAD08		
SEXTUPOL.022.195	0.3	0.0	500	S	6POL02 -- PLEASE KEEP IT "OFF"	...	
TRIM.022.198	0.0	0.0	250	N	TRIM02 Horizontal		

Collimator Status [Collimators]

Beam: H4 / RD51
File: H4A.RD51.034 Momentum: -100.00 GeV/c

Collimators	Read Jaw 1	Read Jaw 2	BeamRef Jaw 1	BeamRef Jaw 2	Min	Max	Info
XCSH.022.06	-17.5	17.5	-20.0	20.0	-40	40	COLL01 Horizont
XCHV.022.13	-20.1	20.1	-20.0	20.0	-45	45	COLL02 Horizont
XCHV.022.13	-20.1	20.0	-20.0	20.0	-45	45	COLL03 Momentu
XCSV.022.17	-40.0	40.0	-40.0	40.0	-40	40	COLL04 Vertical
XCHV.022.19	-44.9	45.0	-45.0	45.0	-45	45	COLL05 Horizont
XCHV.022.19	-44.9	45.0	-45.0	45.0	-45	45	COLL06 Vertical A
XCSV.022.21	-40.0	40.0	-40.0	40.0	-48	43	COLL07 Vertical
XCSV.022.38	-40.1	-38.0	-40.0	-38.0	-45	45	COLL08 Momentu
XCHV.022.45	37.9	39.9	38.0	40.0	-45	45	COLL09 Horizont

H4 Wire Chambers Profiles

Beam: H4 / RD51 Last timing: 30.04.2018 08:16:42
File: H4A.RD51.034 Momentum: -100.00 GeV/c Comment: Muons @0mrad (fm-160421) April 2018

XWCA.022.374 - MWPC03	XWCA.022.408 - XWCA05	XDWC.022.490 - XDWC01	XWCM.022.605	XWCM.022.640 - MWPC09
Counts:37.84 Mean: 0.97	Counts:19.16 Mean: -3 +	Counts:1.577E04 Mean: -	Counts:-0.41 Mean: -30.6	Counts:-0.35 Mean: -11.0
XWCA.022.375 - MWPC04	XWCA.022.409 - XWCA06	XDWC.022.491 - XDWC02	XWCM.022.606	XWCM.022.641 - MWPC10
Counts:44.71 Mean: -0.88	Counts:17.09 Mean: 0.38	Counts:1.559E04 Mean: 1	Counts:0.05 Mean: 408.6	Counts:0.35 Mean: 4.31 +

Figure 15.7: The CESAR GUI

15.3 Logging

The CERN-wide Accelerator Logging Service (CALS) was born out of the LHC Logging Project (a sub-project of the LHC Controls Project) which was mandated in October 2001, based on the experience with LEP [49]. The current mandate can be summarised as:

- Information management for accelerator performance improvement;
- Meet Installation Nucleaire de Base (Basic Nuclear Facility) (INB) requirements for recording beam history;
- Make long-term statistics available for management;
- Avoid duplicate logging efforts.

The scope covers the whole CERN accelerator complex and related infrastructure. Stakeholders are from all over CERN, including representatives of operations teams, equipment groups, and physicists. The acquired data is expected to be persisted online beyond the lifetime of the LHC.

CALS persists time series data coming from pre-defined signals into Oracle databases, and provides an API and a generic application (TIMBER) that can be used to extract and visualise logged data. Conceptually, time series data is persisted and made available for so-called "variables". A variable represents a quantity or status (e.g. coming from signal), and is an abstraction from the underlying implementation of the data acquisition infrastructure (e.g. GM properties, FESA device property fields, FGC properties, WinCC OA data points, Technical Infrastructure Monitoring (TIM) tags, RAMSES data points). This abstraction remains valid over time and across changes in the underlying data acquisition infrastructure. It also serves the end users of the data (which are usually diverse and different from the data acquisition infrastructure developers), by adhering to the CERN-wide Quality Assurance scheme for naming of entities and signals – which helps to find signals (of which there are more than 1.5 million) based on the names, which usually correspond to the physical functional entities that makes up the CERN accelerator complex.

Fundamental data is another logging-specific concept which deals with certain time series data that identifies fundamental events in the accelerator complex – namely at a moment in time, for a given accelerator, for what the beam being used e.g. for cycling machines like the SPS – what was the LSA cycle configuration, what was the timing user and what was the beam destination. This data can then be used to filter normal time series data in order to only retrieve data when certain conditions were met (e.g. SPS beam intensity for 25ns beams sent to LHC). For the LHC, which is not a cycling machine, fundamental data can be considered as the LHC fills – each identified by a unique number, and containing one or more beams modes (e.g. setup, injection, ramp, squeeze, adjust, stable, ramp down).

The current system has proven to be extremely reliable since it was first put into production in September 2003. It is highly available, with almost no downtime over the last 15 years [48, 51]. It is also highly scalable in terms of data ingestion, particularly with respect to initial forecasted data rates of 1TB/year for LHC operation – in 2017 data rates exceeded 2.5TB/day. The current system is relatively simple in terms of number of components, installation, updates etc.

In order to achieve the high performance required, the design and implementation leverages a huge number of highly advanced, Oracle specific optimisations – both in the database itself and Java Database Connectivity (JDBC) code that inserts or extracts data [50, 54]. To maintain or extend the system further requires a very high-level of Oracle expertise – something that, for many years, has been increasingly difficult to recruit for.

Although the system has scaled well in terms of data ingestion and provides linear response times to extract data over time (irrespective of the total stored volume of data), the stabilisation of operating the LHC, combined with the need to prepare for HL-LHC, led to new ways of using the system post-LS1. Certain user communities aim to store a lot more complex data (e.g. so-called bunch-by-bunch and turn-by-turn data stored as large vectors or 2d arrays) to perform regular analysis of such data over extended periods of time (e.g. weeks to months), in order to study new ways to optimise beam configurations and identify the most efficient operational scenarios. Once ingested, the current system struggles to filter such large complex data, and in terms of analysis, is simply too slow to extract the data (e.g. taking up to half day to extract one day of complex data).

Depending on the type of data, it is logged into a short-term storage (speed layer) and/or a long-term storage. The extraction facilities can extract data from both data stores in a manner more or less transparent for the end user. Data can be extracted as logged (i.e. all data for a given signal during a given time window), or pre-filtered and/or aggregated following some common uses cases (e.g. periodic average within a window, aligning data for multiple signals whose raw data was acquired at different times, filtering of data based on values or coincidence with data from a given driving signal).

The first production version of the CALS system was used in September 2003 to capture data from the first TT40 extraction tests (extracting beams from the SPS into the transfer tunnel towards the LHC tunnel). In 2005, the Measurement Database (MDB) was developed to act as a short-term persistence and speed layer for data acquired from CMW accessible devices [53]. Optional transfer of data towards the long-term storage in the Logging Database (LDB) was made available at the same time, offering data-driven, configurable filtering of data based on value changes or elapsed time. Figure 15.8 illustrates the original architecture and how it integrates in the Control System.

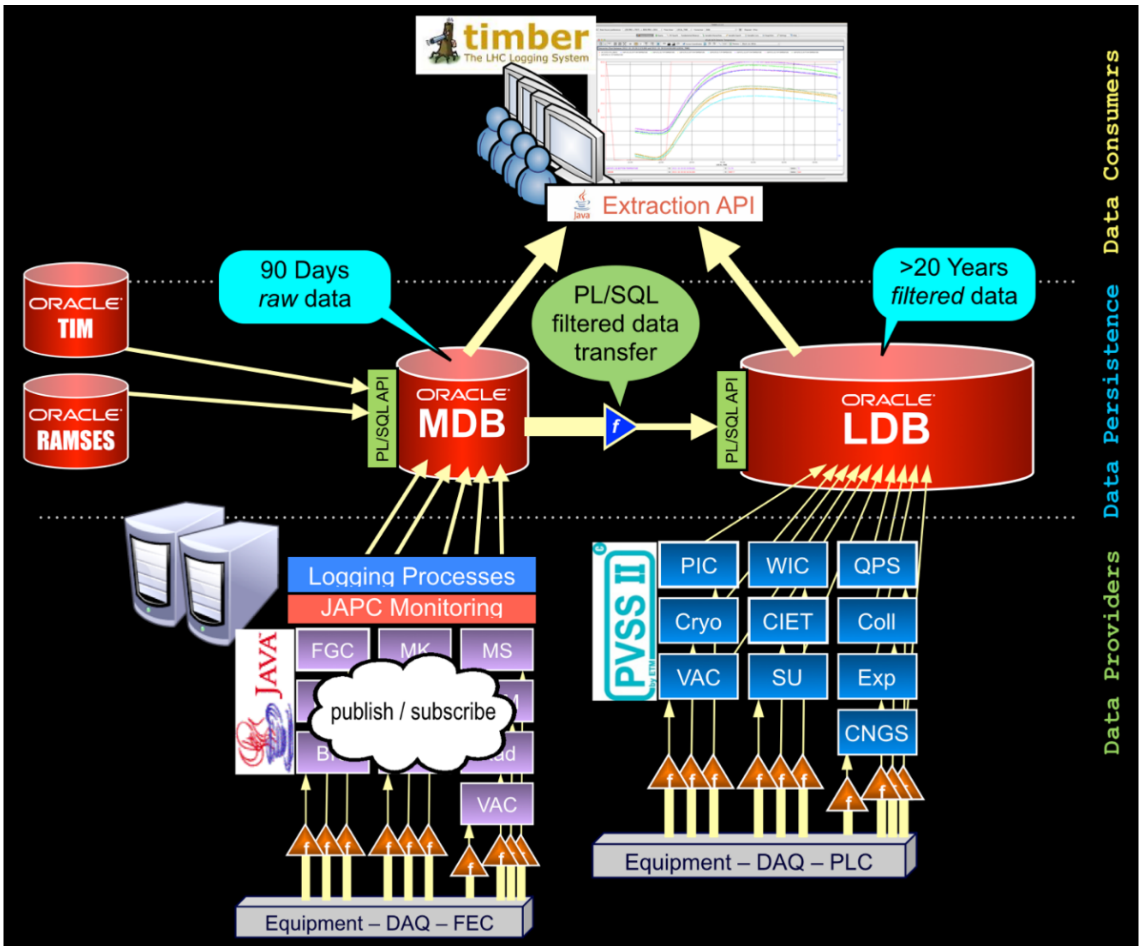


Figure 15.8: CALS architecture overview

In 2012, the CALS system was capturing some 5 billion records per day, and serving approximately 5.5 million user requests per day to extract time series data sets of varying sizes. This was for a user community of more than 1000 people from across CERN, using either TIMBER or one of more than 130 custom applications built on the common extraction API.

In 2017, the CALS system continued to satisfy the needs of many users and its scope continued to expand to cover the needs of new experiments and facilities such as Advanced Proton Driven Plasma Wakefield Acceleration Experiment (AWAKE) and LINAC4. Nevertheless, new ways of using the system post-LS1 have generated additional loads on the internal data processing, and exposed shortcomings with respect to data analysis. As such, during the first half of 2016, a prototype for a new logging system was developed based on modern, open-source, horizontally scalable, Big Data technologies and data science tools. Figure 15.9 depicts the new architecture, along with the main technologies used. The positive results of the prototype work, combined with an inevitable continued increase in data rates and data analysis needs in the near future, led to the official launch of the Next CERN Accelerator Logging Service (NXCALS) project, aiming to deliver a new production system to replace the current CALS system during LS2.

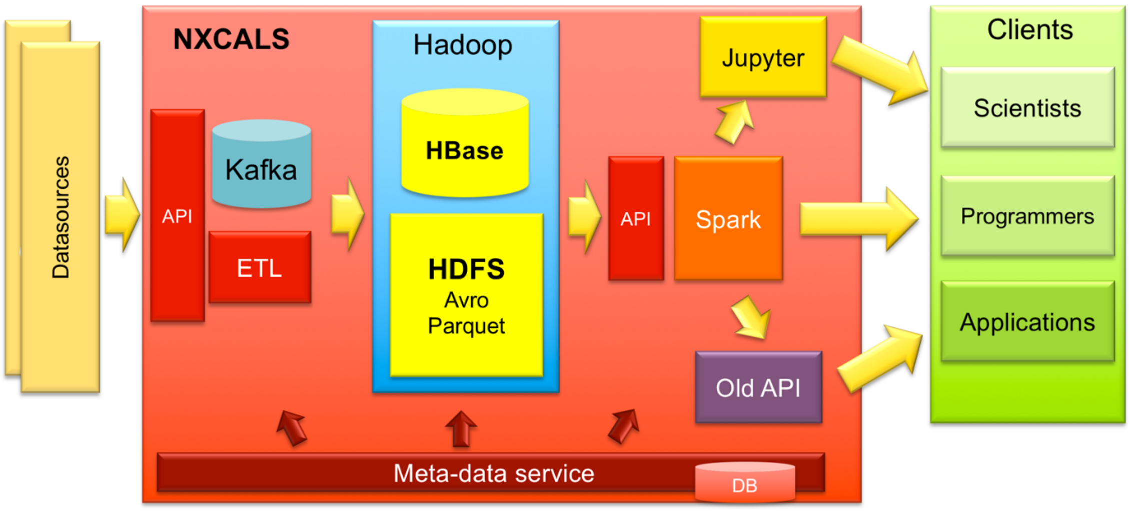


Figure 15.9: NXCALS architecture overview

15.4 Data Concentrators

There are cases where the data published by low-level front-end computers is not directly usable by the various services and applications e.g. CALS (see section 15.3) and Fixed Displays (see section 17.1.3). Typically, some FESA classes expose low-level implementation details or the data is in an inappropriate form (e.g. 2D array) and some post-processing is required. In this type of situation, it is interesting to re-expose the post-processed data using the same device-property paradigm, in order to avoid having application-specific models or transport. Data concentrators were developed in order to acquire device-property data, post-process then republish it via the same means.

As shown in figure 15.10, the typical use-case for concentrators is to consume data sources from different front-end computers that publish data at different moments in time, with no synchronisation. This means that one of the first things to do in a data concentrator is to ensure that we have a coherent set of data. This is done in the ‘event building’ layer by grouping the data streams using acquisition timestamps provided by the timing system. In a cycling machine, only data with the same cyclestamp can be post-processed together. For non-cycling machines, we define other criteria, such as a time window.

Since data concentrators are high-level Control System components, they are by de-facto implemented in Java. The standard method of subscribing to data is with JAPC. As explained in section 14.3, this allows us to access data from different sources (CMW, JMS, Timing library, etc.). Republishing used to be done with JMS but was recently

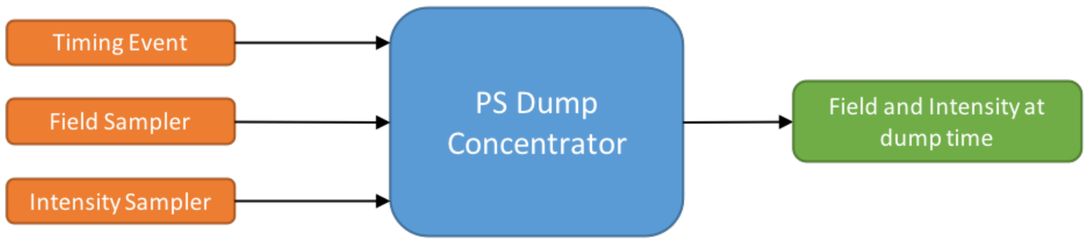


Figure 15.10: PS dump concentrator

moved to CMW-RDA3 in order to profit from its advantages. Due to the large amount of subscriptions that need to be managed, we use a library called JMON (see section 14.3.1) on top of raw JAPC to ease the process. From a deployment point of view, the approach is kept simple; there is just one process per post-processing algorithm.

Originally, as the number of data concentrators was limited ($\sim 10-15$), it was not deemed necessary to invest in developing a framework. This approach has several disadvantages. Firstly, each time a new data concentrator is required, a lot of boilerplate code needs to be repeated and the resulting device-property class needs to be declared manually in the CCDB. Secondly, because there is no framework, non-java experts cannot develop new concentrators easily, therefore the controls group has to participate in their development, despite the fact that the post-processing knowledge belongs to the equipment experts.

In order to overcome some of these shortcomings, the Unified Controls Acquisition and Processing Framework (UCAP) project was launched. It is described in section 15.5. This will allow the post-processing logic to be implemented by the operators and equipment experts, either in Java or languages which are familiar to physicists such as Python. This approach has proven to be successful by the FESA framework.

15.5 Unified Controls Acquisition Processing

The Unified Controls Acquisition Processing (UCAP) project was launched in 2018, aiming to provide a common implementation for a pattern repeated in many core components: acquire, process and publish device-property data. Indeed, Data Concentrators (see section 15.4), InCA acquisition core (see section 15.1.2) and OASIS virtual signals (see chapter 23) are all examples where the component subscribes to several device-properties, synchronises the updates, processes the data and republishes the result as a device-property.

UCAP is intended to replace and extend the functionality of JMON, and provide an easy to use, online analysis tool. It should be possible to setup a new transformation declaratively, by specifying the input parameter and how to synchronise them, the transformation to apply and the device-property model to publish the results. The code for the transformations can be written by the users in several languages (Java, Python, etc.).

The UCAP high-level architecture has three layers, as shown in figure 15.11. At the bottom, the Event Builder subscribes to the source parameters and groups the updates according to the policies defined by the user (e.g. group by cyclestamp). The grouped data is then sent to the transformation to be post-processed, and the top layer handles the publication of the data, as the specified device-property structure, based on CMW-RDA3. In addition, UCAP will provide complete diagnostics with metrics and tools to inspect the input and output flows. It is also foreseen to provide a way to replay events, in order to test the transformation code. UCAP will rely on the Controls Configuration Service (CCS) to register the parameters used by a given transformation, as well as to publish the structure of the result as a so-called virtual device. This will help to improve the support of the Control System's constant evolution by detecting backward incompatible changes, such as the renaming of low-level devices.

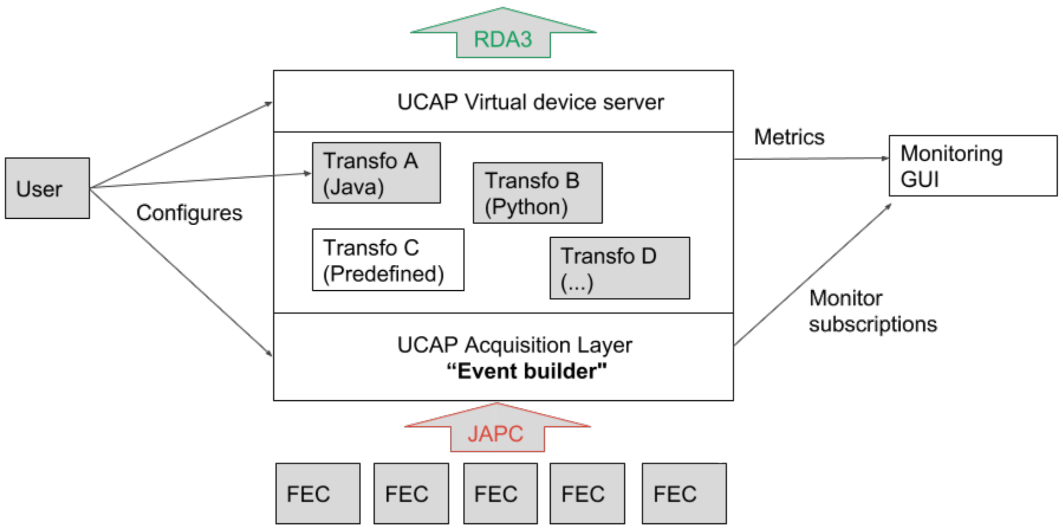


Figure 15.11: UCAP high-level architecture

UCAP code is implemented in Java and relies on modern concepts such as streams, provided by Spring's Reactor Core. One of the features of this implementation is to use UCAP as a library and embed it directly in an application, instead of relying on the server infrastructure. This approach could be used to renovate the Fixed Displays (see section 17.1.3).

The key challenges when providing the UCAP service are to properly isolate the custom transformation code to avoid any ripples in the system if the code fails and to have an infrastructure that will scale properly as increasing numbers of transformations are deployed. A possible solution to the first challenge is to use Function As A Service (FAAS) solutions. For the scalability, the group is currently evaluating industry standard solutions such as Kubernetes, to be able to distribute the individual transformations across several servers.

15.6 Post-Mortem

When operating a machine as complex as the LHC, it is important to be able to understand what happens when there are sudden, unexpected events, such as beam dumps. In such cases, it is important to collect as much information as possible from the different systems in order to be able to analyse the event comprehensively.

The Post-Mortem (PM) system was originally developed for LHC operations [66], but the functional scope of the system has been extended and now covers other situations such as the Injection Quality Check (IQC) [23] and system-specific post-mortems, such as the LHC beam dump system's eXternal Post-Operation Check (XPOC) [42]. In addition, the geographical scope was extended to cover other accelerators such as the SPS.

15.6.1 Data Collection and Storage

The first step in the Post-Mortem chain is distributed across all of the electronic boards and software processes that have the responsibility for gathering their own post-mortem data. This is typically a small rolling buffer of high-frequency data that is frozen upon receiving an event. For example, for the LHC, these events are Post-Mortem 1 and 2 and Dump Ring 1, or Dump Ring 2. Once an event is received, the software processes in the agents collect the data and send it to the central PM server. The Post-Mortem Front-End (PMFE) server stores the data in the fast storage and notifies the PM back-end service, as shown in figure 15.12. In turn, the back-end service triggers the transfer of data from the front-end storage to the larger, but slower back-end storage.

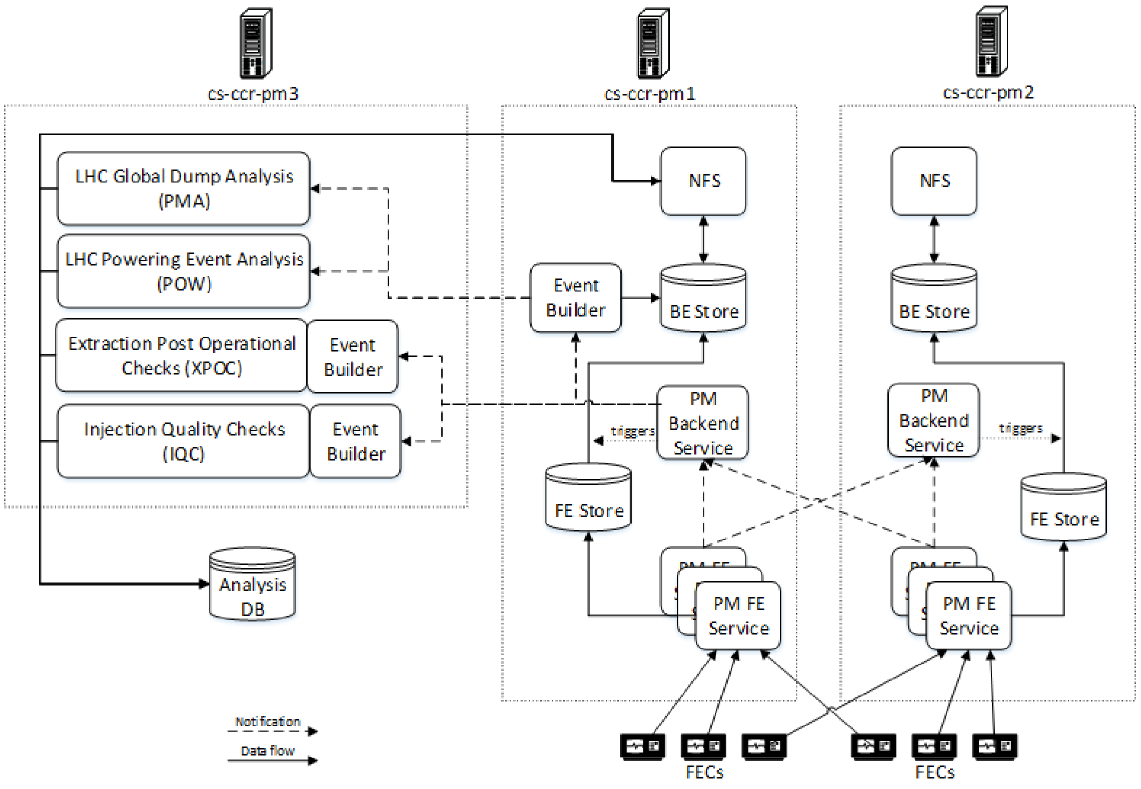


Figure 15.12: Current Post-Mortem architecture

Since it is vital not to lose any PM data, the data transfer between the FECs and the PMFE server cannot rely on loosely coupled mechanisms such as publish-subscribe. Instead, we rely on RDA2 set operations, which guarantees and acknowledges when the data has been written to the FE storage. For scalability reasons, there are multiple PMFE server devices, which receive data through the RDA sets. The device to be used by a given front-end computer will depend on the data domain i.e. BLM, FGC, etc. Although this mechanism does not provide real run-time load balancing, the load is distributed, but statically.

In order to allow fast collection of the data, the PMFE store is a mix of Solid State and Hard Drive Disks with a total capacity of 1TB. On the other hand, the PM back-end store is based on RAID10 and the ext4 journalised file system. It can store up to 11TB of Post Mortem data. The data is stored in files as serialised CORBA objects, as this is the underlying technology for CMW-RDA2.

The data published by the FECs is structured in three levels: system, class and source. The system and class define the data schema and therefore cannot be changed without breaking the analysis modules. Examples of systems include BLM, FGC and their corresponding classes could be BLMLHC, FGC51, etc. The systems are defined and assigned by the PM team in collaboration with their users. Typical sources are the names of the front-end devices providing the data. In addition to the data, there is a data qualifier (e.g. test, incomplete), a timestamp and other meta-data, describing units or array mappings, amongst others. The data is stored as it is received, without any modifications, but upon being read, it can be adapted to take into account new versions or to fix problems. With this system-class-source data structure, a given front-end device can send data for more than one system-class.

The next step in the Post-Mortem processing is the event building, which ensures that all of the data required for the processing is present and coherent. There are several event builders depending on the specific Post-Mortem instance, but the default one works as follows: Upon reception of the first data, a collection window is opened using the data timestamp t_0. By default, the collection window spans a range from t_0-100ms to t_0+800ms. The window stays open for a period of 12 minutes to allow all of the relevant data to be collected, and any data received during this time with a timestamp in the collection window becomes part of the event. Once the event building is complete, an event file is persisted in the storage, containing references to the different data files. If during the collection period, another piece of data is received with a timestamp falling outside of the current collection window, a new collection window is opened and the processes are maintained in parallel.

The vast majority of the data used to build events comes from the post-mortem storage, however in a few cases the event builder also takes data from JAPC parameters, such as timing data that is not pushed into the post-mortem storage.

The 12-minutes of data gathering may appear excessive, but some systems are slow to provide data. Therefore, in order to provide faster feedback to the operators, a pre-analysis is triggered after 2 minutes, as by this time the majority of the required data will be present. Nevertheless, the latency of the current PM system limits the performance of some operations, such as the injection and dump. In an ideal world, this would be performed for every SPS cycle i.e. every 10 seconds, but due to latency, it can only be performed every 2 minutes.

15.6.2 Data Analysis

The Post-Mortem Analysis (PMA) framework [31] executes analysis modules that are usually developed by the equipment experts or operations team. The modules are organised as a graph, as shown in figure 15.13.

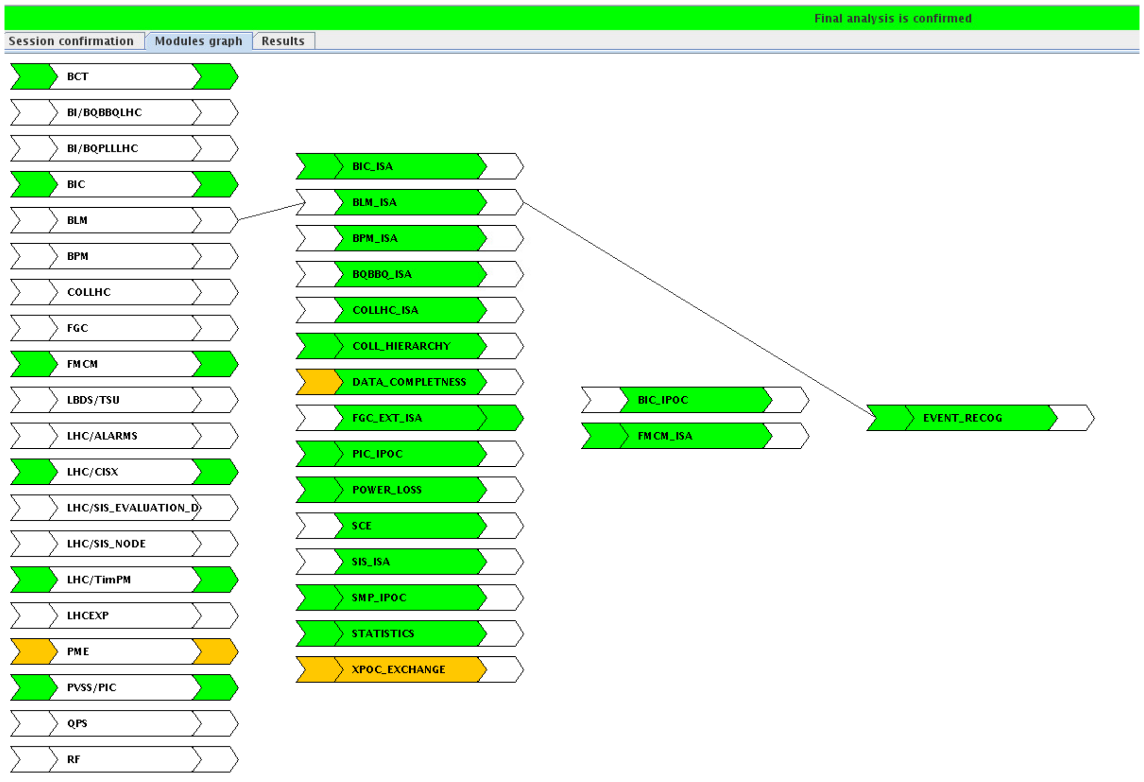


Figure 15.13: Graph layout of Post-Mortem analysis modules

The current PMA system contains approximately 50 analysis modules of varying complexity. An analysis module is a Java class that implements a specific interface. In order to ease the development of the modules, Java Beans are generated to represent the incoming and outgoing data. In addition, the framework provides typical features to ensure data integrity and ease debugging, such as a graphical view. The PM analysis system also allows an analysis of an event to be replayed on an offline system, thus allowing the developer to develop and debug his analysis code or to use additional data by extending the collection window.

The execution is triggered by the event-builder module and uses data directly from the PM storage. In turn, the results of the analyses are sent back to the PM storage and if necessary they can also be exported to SIS (see section 16.2) or sent as a report by email.

The Post-Mortem system is the only event-based analysis solution in the Control System, and it is being applied to more and more use cases. However, the current architecture, initially developed in 2008, is starting to show its limitations. The implementation based on NFS exposing serialised CORBA objects in files has leaked into the clients and the RDA2 sets are synchronous calls, blocking calling threads. The analysis part depends on JMS and is implemented as a monolithic structure, and as such, does not scale well horizontally.

In the near future, the data collection and storage will be refactored based on the solution put in place for NXCALS. The PM front-end service will be replaced by a set of data collectors that will send the data to NXCALS through its ingestion API. However, Post-

Mortem system requires faster access to the collected data than the latency foreseen by the initial NXCALS design, for this reason it is necessary for the Analysis Service and Event Builder to access NXCALs data through a low-latency access point, as depicted in figure 15.14. In addition, the first version of the data collectors will be based on RDA3, but further study is required in order to improve the load-balancing capabilities of either RDA3 or another specific solution [30].

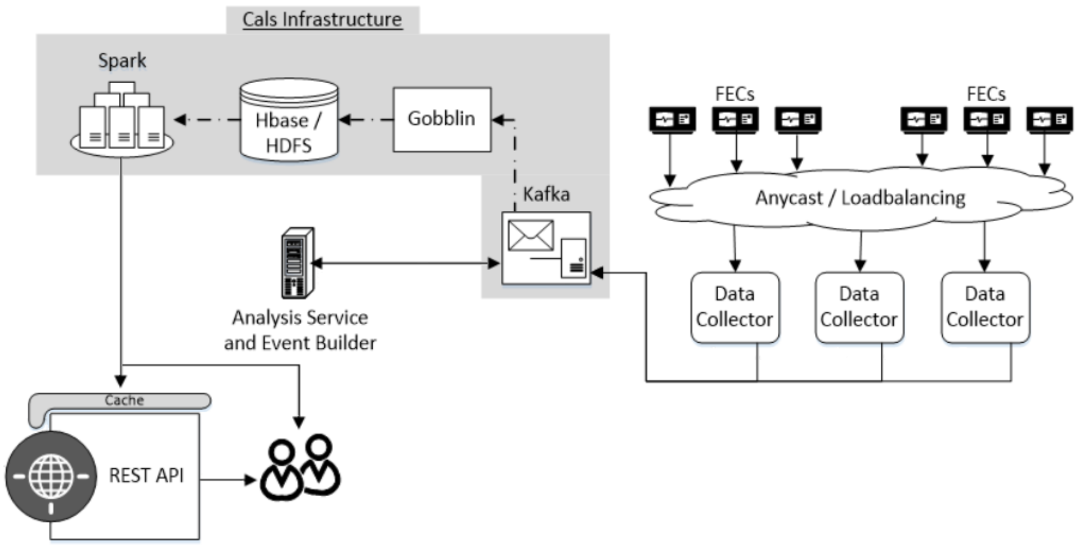


Figure 15.14: New architecture of the Post-Mortem system

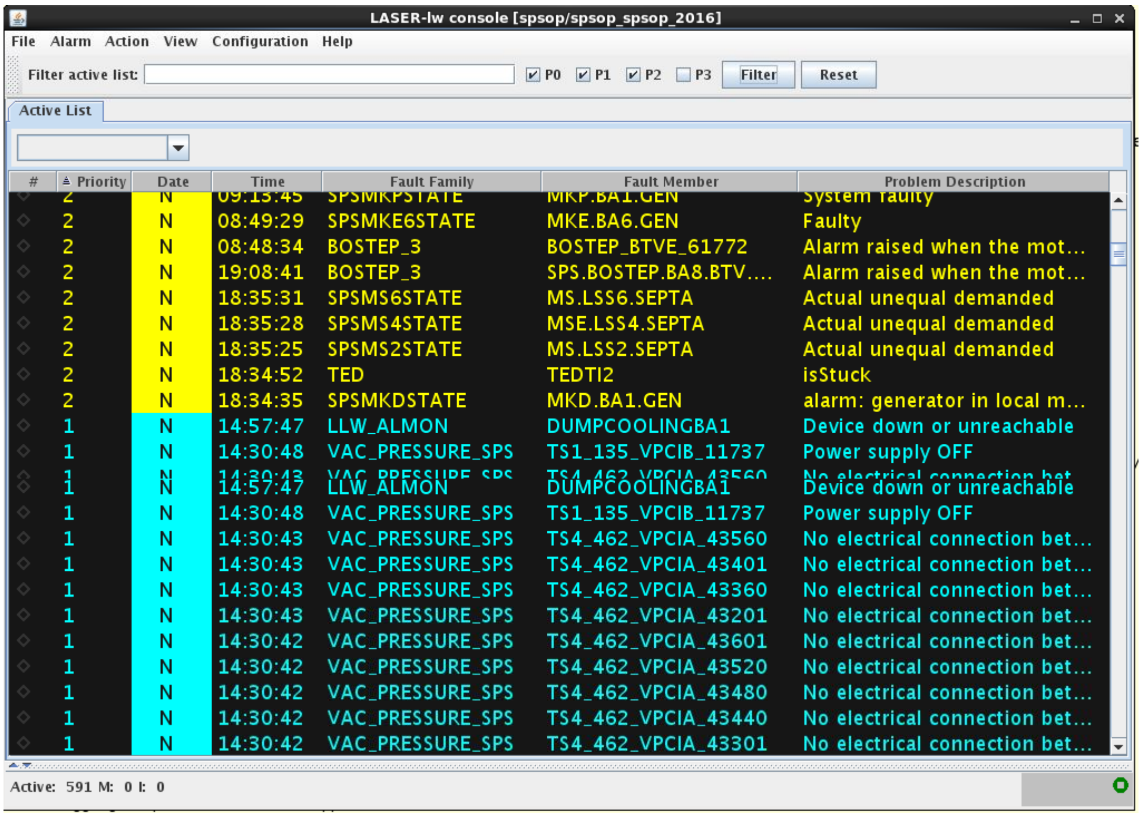
On the analysis side, the long-term plan is to allow analysis modules to be deployed independently. This will solve two common problems with our high-level systems, firstly, that deploying a single change requires a complete redeployment, and secondly, external contributions that do not work properly cannot easily be isolated.

15.7 Alarms

An accelerator is made of many sub-systems that can fail and require interventions from the operators. The information that something requires attention is conveyed through alarms. LHC Alarms Service (LASER) is the system that handles alarms for both accelerators and the Technical Infrastructure (TI). Today, LASER handles 350,000 potential alarms.

TI alarms cover subsystems such as electricity distribution, cooling and ventilation etc. and the operators handle problems in a very classical way [60]. Alarms are displayed until the operators acknowledge them. They are prioritised, and introducing a new alarm requires formal documentation. On the other hand, for the accelerator operators, alarm handling is more flexible and alarms are automatically acknowledged if the error condition disappears. The accelerator alarms can be active on one cycle, but not another; therefore, the alarm system needs to be aware that the accelerators are multiplexed.

In LASER, the subsystems producing alarms are called alarm sources and an individual alarm is identified by a triplet of fault family, fault member and fault code. The alarm sources periodically send an alarm message to the LASER server, describing any alarm that was activated or terminated since the previous message. Today, this is based on JMS. On the server-side, alarm definitions are fetched from the CCDB and the basic message is augmented with additional data such as the alarm priority and description. Then, the alarms are grouped by alarm category e.g. PSB operations, before being sent to the Graphical User Interfaces (Alarm console) subscribed to the given alarm categories. On the alarm console, as depicted in figure 15.15, the alarms are displayed, sorted by priority, in order to ensure that the most urgent action is taken first. Another element of the LASER system subscribes to all of the categories and archives the alarms, in order to keep a history. For the accelerator domain, the biggest alarm source comes from the controls devices. To avoid introducing a new concept in parallel to the device-property model, the device's alarms are exported through a specific property. Alarm Monitor processes subscribe to the alarm properties and transform property notifications into alarm messages.



#	Priority	Date	Time	Fault Family	Fault Member	Problem Description
	2	N	09:13:45	SPSMKPSTATE	MKP.BA1.GEN	System faulty
	2	N	08:49:29	SPSMKE6STATE	MKE.BA6.GEN	Faulty
	2	N	08:48:34	BOSTEP_3	BOSTEP_BTVE_61772	Alarm raised when the mot...
	2	N	19:08:41	BOSTEP_3	SPS.BOSTEP.BA8.BTV....	Alarm raised when the mot...
	2	N	18:35:31	SPSMS6STATE	MS.LSS6.SEPTA	Actual unequal demanded
	2	N	18:35:28	SPSMS4STATE	MSE.LSS4.SEPTA	Actual unequal demanded
	2	N	18:35:25	SPSMS2STATE	MS.LSS2.SEPTA	Actual unequal demanded
	2	N	18:34:52	TED	TEDTI2	isStuck
	2	N	18:34:35	SPSMKDSTATE	MKD.BA1.GEN	alarm: generator in local m...
	1	N	14:57:47	LLW_ALMON	DUMPCOOLINGBA1	Device down or unreachable
	1	N	14:30:48	VAC_PRESSURE_SPS	TS1_135_VPCIB_11737	Power supply OFF
	1	N	14:30:43	VAC_PRESSURE_SPS	TS4_462_VPCIA_43560	No electrical connection bet
	1	N	14:57:47	LLW_ALMON	DUMPCOOLINGBA1	Device down or unreachable
	1	N	14:30:48	VAC_PRESSURE_SPS	TS1_135_VPCIB_11737	Power supply OFF
	1	N	14:30:43	VAC_PRESSURE_SPS	TS4_462_VPCIA_43560	No electrical connection bet...
	1	N	14:30:43	VAC_PRESSURE_SPS	TS4_462_VPCIA_43401	No electrical connection bet...
	1	N	14:30:43	VAC_PRESSURE_SPS	TS4_462_VPCIA_43360	No electrical connection bet...
	1	N	14:30:43	VAC_PRESSURE_SPS	TS4_462_VPCIA_43201	No electrical connection bet...
	1	N	14:30:42	VAC_PRESSURE_SPS	TS4_462_VPCIA_43601	No electrical connection bet...
	1	N	14:30:42	VAC_PRESSURE_SPS	TS4_462_VPCIA_43520	No electrical connection bet...
	1	N	14:30:42	VAC_PRESSURE_SPS	TS4_462_VPCIA_43480	No electrical connection bet...
	1	N	14:30:42	VAC_PRESSURE_SPS	TS4_462_VPCIA_43440	No electrical connection bet...
	1	N	14:30:42	VAC_PRESSURE_SPS	TS4_462_VPCIA_43301	No electrical connection bet...

Active: 591 M: 0 I: 0

Figure 15.15: Alarms console

Two challenges of alarm systems are reliability and avalanche control. The alarms system is particularly required when major faults occur, therefore the alarm system must not be built upon the same infrastructure as the systems it supervises. In order to avoid dependencies on the Controls Configuration Database (CCDB), the alarm definitions are extracted and stored in an offline XML file. Likewise, the archiving process is decoupled from the main process, thanks to JMS, and furthermore if it cannot store the new alarms, it falls back on a local text file.

Also, when a major event occurs, many systems will produce alarms, a so-called avalanche, and the system must be designed to handle this type of situation smoothly. In addition, the system has to take into account that some alarm sources can fail and start flooding the server with alarm messages. The strategy in LASER is to introduce flood-control at the source level and ensuring that the server can handle a predefined number of messages, typically 250 messages per minute.

Before the LASER project was launched in 2001 [13, 34], there was a system called the CERN Alarm System (CAS) that had been developed in the 90's. The first version of LASER was based on technologies of the time such as J2EE and Oracle OC4J. Since then, the system has been simplified and is based on more modern components to ease maintenance. Figure 15.16 shows the current architecture.

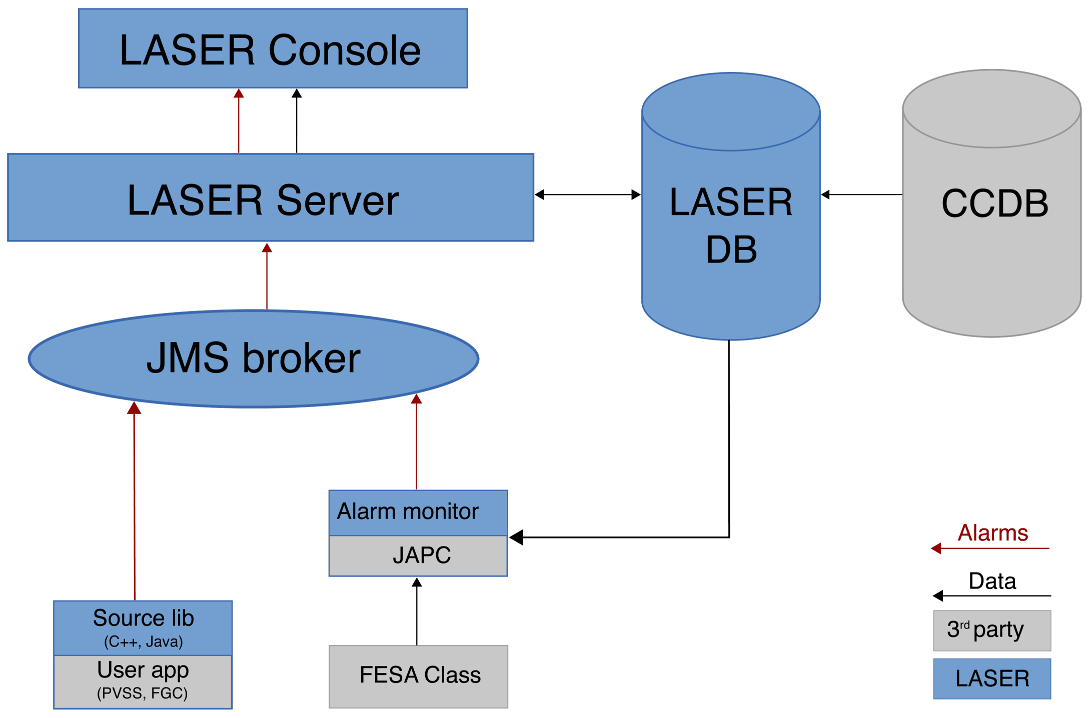


Figure 15.16: LASER architecture

As described during the introduction, there is a big difference between the way that the TI alarms and the accelerator alarms are handled by their respective operators, and therefore, in 2017, it was decided to split the two domains and have a specific system for each. From now on, LASER-lw will handle accelerator alarms only, and CERN Control and Monitoring Platform (C2MON), whose description is outside the scope of this document, deals with the TI alarms.

16. Automation

The low-level Control System provides the means to automate processing as described in chapter 11. At that level, the timing events are the most common source of triggers and the control is normally limited to the local FEC. There are two main use cases where we want to automate the actions in the Control System at a higher level. The first one is event-driven and works very much like the FEC case but using distributed sources and actions; this is a scenario where the Software Interlock System, as described in section 16.2, can be used. The second case is typical of huge machines, such as LHC, where the number of actions to be performed can be daunting (hundreds or even thousands of steps). What is needed in these situations is a tool that performs a suite of actions in a repeatable manner therefore removing manual, error-prone tasks from the accelerator operators; this is a job for the Sequencers.

16.1 Sequencers

The Controls group provides two different sequencers for two different purposes. The Hardware Commissioning Sequencer (HWC) for automating hardware tests [6] and the Beam Operation Sequencer for automating beam operations in the different accelerators such as the LHC, SPS, and LEIR.

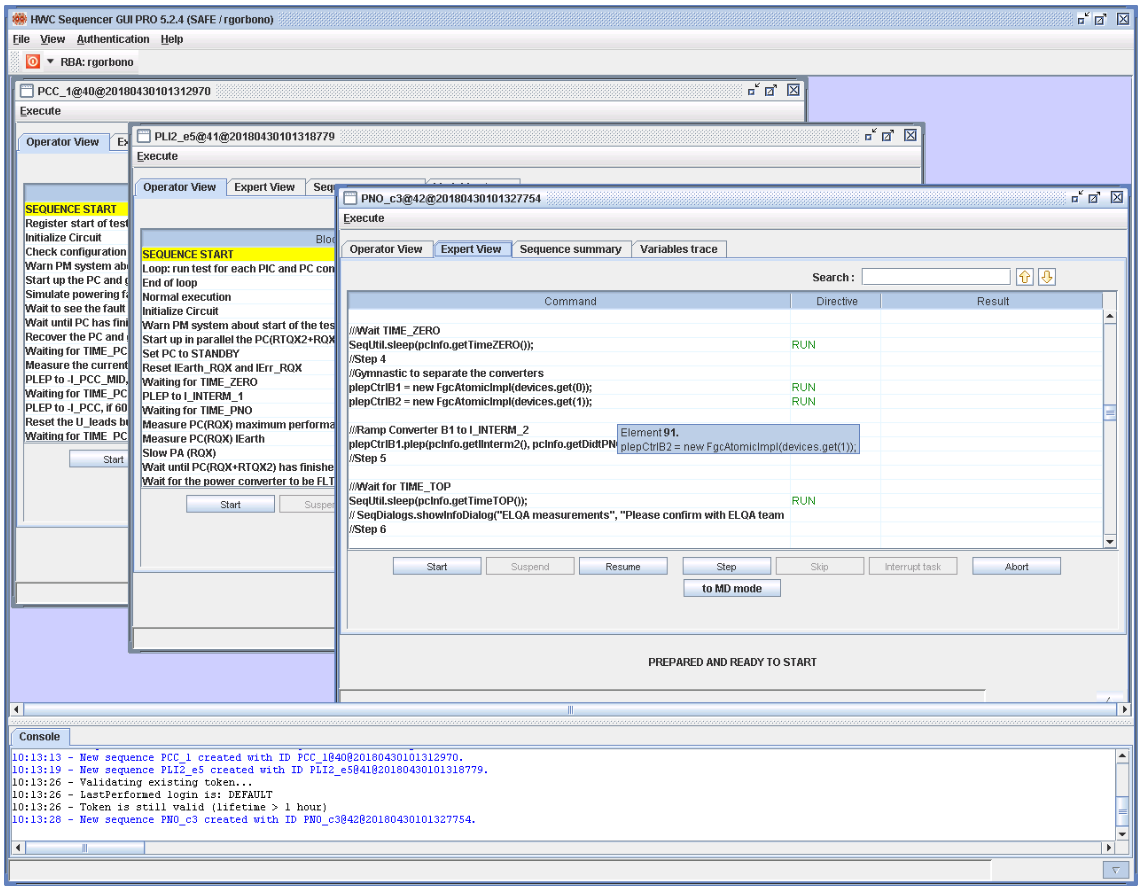
At the heart of the sequencers, there is the concept of a task. A task interacts with the accelerator equipment, typically by setting a new value, followed by a read operation to either check the hardware or wait for a condition. For more complex operations, one can assemble several tasks into a macro task. In addition to automating common operations, the sequencers behave like a debugger, allowing the operators to set breakpoints and skip tasks. As not respecting the designed sequence may require more advanced knowledge and jumps can create problematic situations, different modes exist; a safe mode without jumps, and a MD mode with all the options, the latter being the current default.

As most of our high-level software, the sequencers are implemented in Java and therefore, we can profit from the usual tools, such as Version Control Systems (VCSs).

16.1.1 Hardware Commissioning Sequencer

The hardware commissioning team focuses on the validation of the LHC electrical circuits and for that purpose, they need a tool to write and execute test suites as sequences of actions in a given order. The execution of the test sequences needs to have some flexibility, such as flow-control i.e. if and loop statements. The Hardware Commissioning Sequencer has a fail-fast approach in the sense that the running sequence is terminated on error. Due to the limited complexity of the tests, only a one-level sequence is required.

Following the requirements and taking into account the limitations of Java, the sequencer is based on Pnuts, a Java-compatible scripting language. Indeed, Java does not allow the possibility to skip statements and only provides breakpoints. To stay with known and supported technologies, the sequences are written in Java and either transformed into Pnuts scripts for production or executed directly in the Eclipse IDE at development time. This possibility is a clear advantage of using Java, as it eases the development and debugging of sequences. Figure 16.1 shows an example of the Hardware Commissioning Sequencer GUI.


HWC Sequencer GUI PRO 5.2.4 (SAFE / rgorbono)
File View Authentication Help
RBA: rgorbono
PCC_1@40@20180430101312970
Execute
Operator View
PLI2_e5@41@20180430101318779
Operator View
Expert View
PNO_c3@42@20180430101327754
Operator View
Expert View
Sequence summary
Variables trace
Search :
Start
Suspend
Resume
Step
Skip
Interrupt task
Abort
to MD mode
PREPARED AND READY TO START
Console

```
SEQUENCE START
Register start of test
Initialize Circuit
Check configuration
Warn PM system ab
Start up the PC and g
Simulate powering f
Wait to see the fault
Wait until PC has fini
Recover the PC and g
Waiting for TIME_PC
Measure the current
PLEP to -I_PCC_MID,
Waiting for TIME_PC
PLEP to -I_PCC, if 60
Reset the U_leads bu
Waiting for TIME_PC
```

```
SEQUENCE START
Loop: run test for each PIC and PC con
End of loop
Normal execution
Initialize Circuit
Warn PM system about start of the tes
Start up in parallel the PC(RTQX2+RQX
Set PC to STANDBY
Reset IEarth_RQX and IErr_RQX
Waiting for TIME_ZERO
PLEP to I_INTERM_1
Waiting for TIME_PNO
Measure PC(RQX) maximum performa
Measure PC(RQX) IEarth
Slow PA (RQX)
Wait until PC(RQX+RTQX2) has finishe
Wait for the power converter to be FLT
```

Command	Directive	Result
///Wait TIME_ZERO		
SeqUtil.sleep(pcInfo.getTimeZERO());	RUN	
//Step 4		
//Gymnastic to separate the converters		
plepCtrlB1 = new FgcAtomicImpl(devices.get(0));	RUN	
plepCtrlB2 = new FgcAtomicImpl(devices.get(1));	RUN	
///Ramp Converter B1 to I_INTERM_2		
plepCtrlB1.plep(pcInfo.getIInterm2(), pcInfo.getDidtPN		
//Step 5		
///Wait for TIME_TOP		
SeqUtil.sleep(pcInfo.getTimeTOP());	RUN	
// SeqDialogs.showInfoDialog("ELQA measurements", "Please confirm with ELQA team		
//Step 6		

Element 91.
plepCtrlB2 = new FgcAtomicImpl(devices.get(1));

```
10:13:13 - New sequence PCC_1 created with ID PCC_1@40@20180430101312970.
10:13:19 - New sequence PLI2_e5 created with ID PLI2_e5@41@20180430101318779.
10:13:26 - Validating existing token...
10:13:26 - LastPerformed login is: DEFAULT
10:13:26 - Token is still valid (lifetime > 1 hour)
10:13:28 - New sequence PNO_c3 created with ID PNO_c3@42@20180430101327754.
```

Figure 16.1: Hardware Commissioning Sequencer

16.1.2 Beam Operations Sequencer

This sequencer was initially created to drive LHC operations but since then it has been extended to the injectors, where it is used to automate routines that are common to several cycles.

In the case of the LHC Beam Sequencer, there is no need for control statements at the level of the sequences, just a list of actions to execute in order. On the other hand, a nominal LHC sequence can have thousands of steps. With such a complexity, n-levels of sub-sequences may be required, as well as the ability to group and reuse routines. Therefore, a structure made of sequences, sub-sequences, and tasks was developed. Another new requirement was the possibility to execute tasks in parallel and to be able to jump forwards and backwards in a sequence. Figure 16.2 displays part of the "LHC Nominal" sequence. Unlike the Hardware Commissioning Sequencer, the LHC version is not fail-fast. If an error occurs, the sequencer stops and waits for the operator to either fix the problem or skip the task.

When creating tasks, one has to bear in mind that sequences and tasks cannot exchange information between themselves, as they do not return values. However, sequences and tasks can receive input arguments from higher level sequences. In addition, there are limitations that have to be taken into account when choosing between macro-tasks and sub-sequences for more complex operations as, for example, sub-sequences do not have flow control.

As the Pnuts engine could not be sufficiently extended to cover all of our requirements, the LHC Beam Sequencer is based on a home-made Java engine. Unfortunately, this new implementation cannot be reused for the Hardware Commissioning Sequencer, as some features, such as loops, are not available.

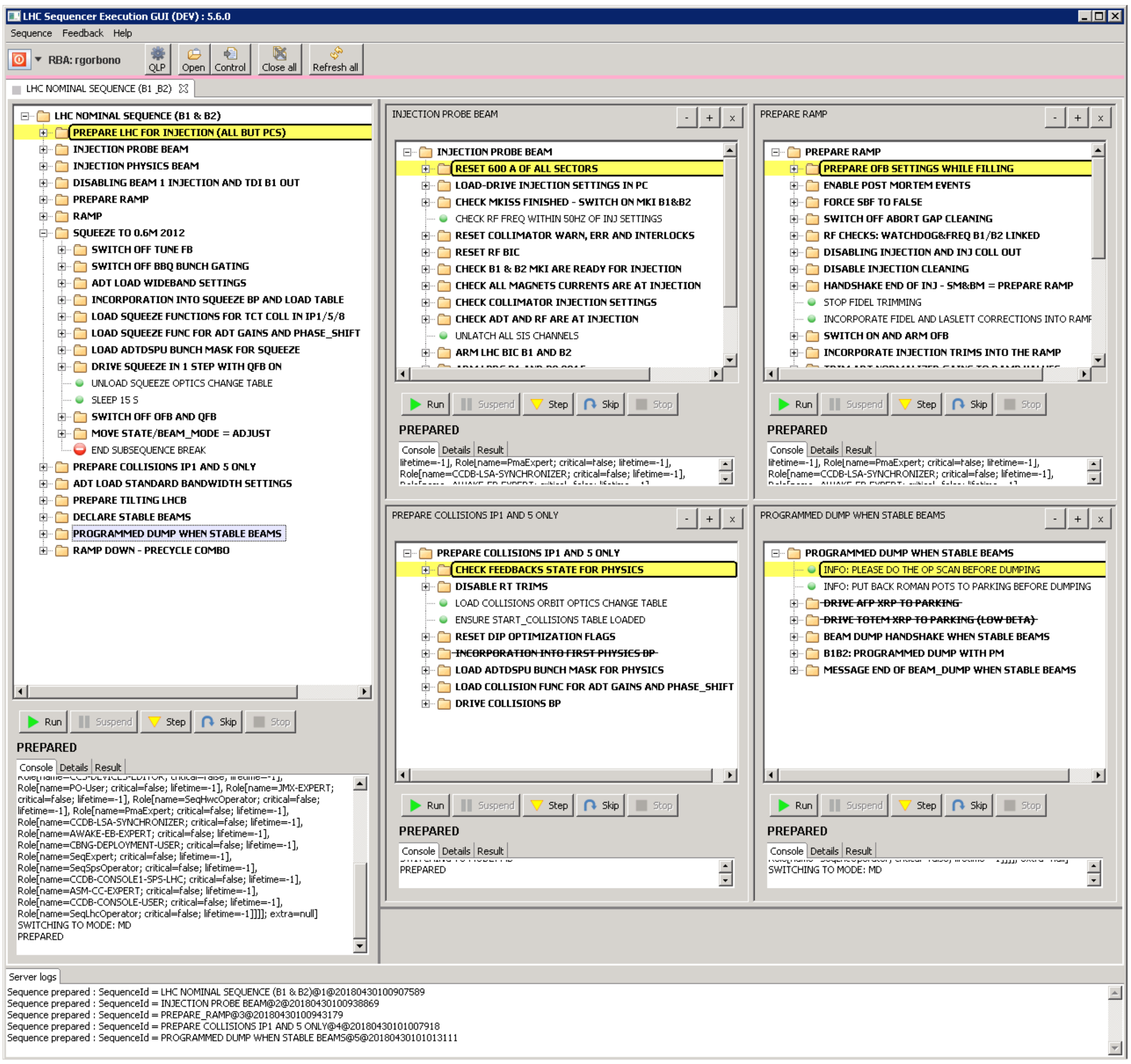
The operators are in charge of creating the sequences, as they have the best global view of the accelerator. On the other hand, the individual tasks are developed by the experts from the equipment groups. For example, the BE-BI group implements specific tasks for their equipment.

The operators use an editor for composing sequences, without needing to write any code. Tasks are displayed as blocks, which can be dragged and dropped into sequences. The editor also provides value completion, advanced editing and validation. Behind the scenes, it generates Java code required to call the tasks, which are Java methods with annotations.

16.2 SIS

The LHC is a complex and expensive machine, and hardware and personnel need to be protected. The Beam Interlock System (BIS) is a failsafe hardware interlock system, with hard real-time constraints. The Software Interlock System (SIS) was developed as a complement to the BIS for configuring higher-level interlocks in a more flexible way, without the need for additional physical hardware or cables. SIS covers the need to react to inputs from the Control System with custom logic that can be implemented by operators.

SIS is more focused on ensuring operational efficiency, rather than machine protection. SIS not only acts as an interlock system during beam operations, but also as a tool to


LHC Sequencer Execution GUI (DEV) : 5.6.0
Sequence Feedback Help
RBA: rgorbono
QLP Open Control Close all Refresh all
LHC NOMINAL SEQUENCE (B1 _B2)
LHC NOMINAL SEQUENCE (B1 & B2)
PREPARE LHC FOR INJECTION (ALL BUT PCS)
INJECTION PROBE BEAM
INJECTION PHYSICS BEAM
DISABLING BEAM 1 INJECTION AND TDI B1 OUT
PREPARE RAMP
RAMP
SQUEEZE TO 0.6M 2012
SWITCH OFF TUNE FB
SWITCH OFF BBQ BUNCH GATING
ADT LOAD WIDEBAND SETTINGS
INCORPORATION INTO SQUEEZE BP AND LOAD TABLE
LOAD SQUEEZE FUNCTIONS FOR TCT COLL IN IP1/5/8
LOAD SQUEEZE FUNC FOR ADT GAINS AND PHASE_SHIFT
LOAD ADTDSPU BUNCH MASK FOR SQUEEZE
DRIVE SQUEEZE IN 1 STEP WITH QFB ON
UNLOAD SQUEEZE OPTICS CHANGE TABLE
SLEEP 15 S
SWITCH OFF OFB AND QFB
MOVE STATE/BEAM_MODE = ADJUST
END SUBSEQUENCE BREAK
PREPARE COLLISIONS IP1 AND 5 ONLY
ADT LOAD STANDARD BANDWIDTH SETTINGS
PREPARE TILTING LHCB
DECLARE STABLE BEAMS
PROGRAMMED DUMP WHEN STABLE BEAMS
RAMP DOWN - PRECYCLE COMBO
Run Suspend Step Skip Stop
PREPARED
Console Details Result
INJECTION PROBE BEAM
RESET 600 A OF ALL SECTORS
LOAD-DRIVE INJECTION SETTINGS IN PC
CHECK MKISS FINISHED - SWITCH ON MKI B1&B2
CHECK RF FREQ WITHIN 50HZ OF INJ SETTINGS
RESET COLLIMATOR WARN, ERR AND INTERLOCKS
RESET RF BIC
CHECK B1 & B2 MKI ARE READY FOR INJECTION
CHECK ALL MAGNETS CURRENTS ARE AT INJECTION
CHECK COLLIMATOR INJECTION SETTINGS
CHECK ADT AND RF ARE AT INJECTION
UNLATCH ALL SIS CHANNELS
ARM LHC BIC B1 AND B2
PREPARE RAMP
PREPARE OFB SETTINGS WHILE FILLING
ENABLE POST MORTEM EVENTS
FORCE SBF TO FALSE
SWITCH OFF ABORT GAP CLEANING
RF CHECKS: WATCHDOG&FREQ B1/B2 LINKED
DISABLING INJECTION AND INJ COLL OUT
DISABLE INJECTION CLEANING
HANDSHAKE END OF INJ - SM&BM = PREPARE RAMP
STOP FIDEL TRIMMING
INCORPORATE FIDEL AND LASLETT CORRECTIONS INTO RAMP
SWITCH ON AND ARM OFB
INCORPORATE INJECTION TRIMS INTO THE RAMP
PREPARE COLLISIONS IP1 AND 5 ONLY
CHECK FEEDBACKS STATE FOR PHYSICS
DISABLE RT TRIMS
LOAD COLLISIONS ORBIT OPTICS CHANGE TABLE
ENSURE START_COLLISIONS TABLE LOADED
RESET DIP OPTIMIZATION FLAGS
~~INCORPORATION INTO FIRST PHYSICS BP~~
LOAD ADTDSPU BUNCH MASK FOR PHYSICS
LOAD COLLISION FUNC FOR ADT GAINS AND PHASE_SHIFT
DRIVE COLLISIONS BP
PROGRAMMED DUMP WHEN STABLE BEAMS
INFO: PLEASE DO THE OP SCAN BEFORE DUMPING
INFO: PUT BACK ROMAN POTS TO PARKING BEFORE DUMPING
~~DRIVE AFP XRP TO PARKING~~
~~DRIVE TOTEM XRP TO PARKING (LOW BETA)~~
BEAM DUMP HANDSHAKE WHEN STABLE BEAMS
B1B2: PROGRAMMED DUMP WITH PM
MESSAGE END OF BEAM_DUMP WHEN STABLE BEAMS
Server logs

```
Role[name=PO-User; critical=false; lifetime=-1], Role[name=JMX-EXPERT;
critical=false; lifetime=-1], Role[name=SeqHwcOperator; critical=false;
lifetime=-1], Role[name=PmaExpert; critical=false; lifetime=-1],
Role[name=CCDB-LSA-SYNCHRONIZER; critical=false; lifetime=-1],
Role[name=AWAKE-EB-EXPERT; critical=false; lifetime=-1],
Role[name=CBNG-DEPLOYMENT-USER; critical=false; lifetime=-1],
Role[name=SeqExpert; critical=false; lifetime=-1],
Role[name=SeqSpsOperator; critical=false; lifetime=-1],
Role[name=CCDB-CONSOLE1-SPS-LHC; critical=false; lifetime=-1],
Role[name=ASM-CC-EXPERT; critical=false; lifetime=-1],
Role[name=CCDB-CONSOLE-USER; critical=false; lifetime=-1],
Role[name=SeqLhcOperator; critical=false; lifetime=-1]]]]; extra=null]
SWITCHING TO MODE: MD
PREPARED
```

```
lifetime=-1], Role[name=PmaExpert; critical=false; lifetime=-1],
Role[name=CCDB-LSA-SYNCHRONIZER; critical=false; lifetime=-1],
```

```
PREPARED
```

```
SWITCHING TO MODE: MD
```

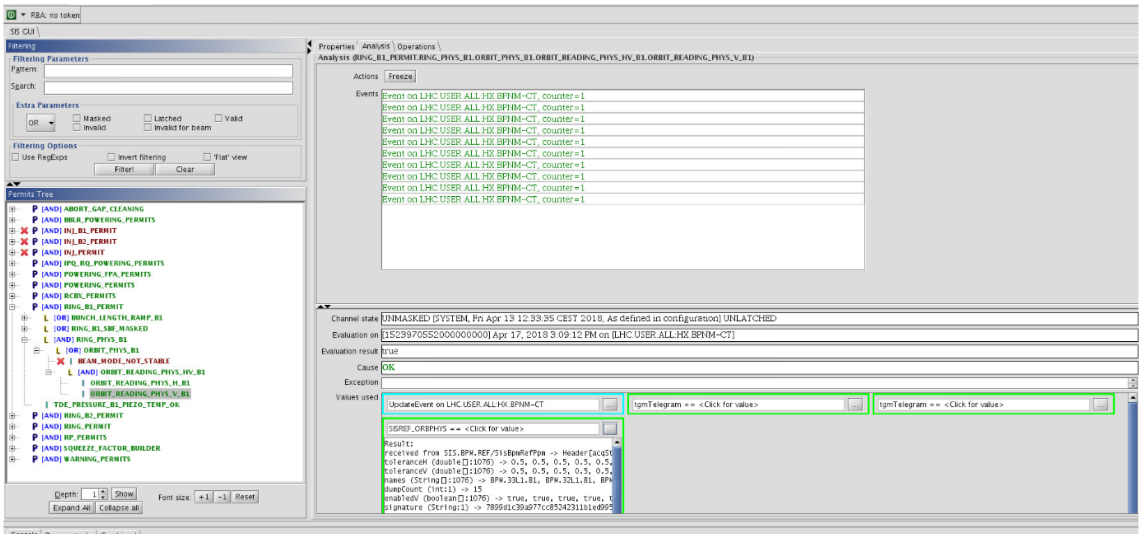
```
Sequence prepared : SequenceId = LHC NOMINAL SEQUENCE (B1 & B2)@1@20180430100907589
Sequence prepared : SequenceId = INJECTION PROBE BEAM@2@20180430100938869
Sequence prepared : SequenceId = PREPARE_RAMP@3@20180430100943179
Sequence prepared : SequenceId = PREPARE COLLISIONS IP1 AND 5 ONLY@4@20180430101007918
Sequence prepared : SequenceId = PROGRAMMED DUMP WHEN STABLE BEAMS@5@20180430101013111
```

Figure 16.2: LHC Sequencer execution GUI

ensure that operational conditions are met before injection.

The core concept in SIS is the Permit. A permit can be considered as a tree where nodes are logical expressions. The left-hand side of figure 16.3 shows a permit tree for Ring 1 of the LHC.

Leaf nodes or Individual Software Interlock Channels (ISICs) are usually simple logical expressions taking values from devices in the Control System e.g. current of Power Converter abc > 10 amps. The intermediate nodes are called Logical Software Interlock Channels (LSICs), which combine Boolean results from the lower-level nodes. On every node, one can attach exporters that perform certain actions based on the evaluation of the node, such as creating an entry in an e-logbook, trimming a parameter, activating or terminating an alarm etc. Since any node can have exporters, it is important that all of the nodes are always evaluated. Indeed, typical optimisations to skip node evaluation, such as when the upper node performs an OR operation and the final result is already known, cannot be applied here.



```
P [AND] ABORT_GAP_CLEANING
P [AND] BBLR_POWERING_PERMITS
P [AND] INJ_B1_PERMIT
P [AND] INJ_B2_PERMIT
P [AND] INJ_PERMIT
P [AND] IPQ_RQ_POWERING_PERMITS
P [AND] POWERING_FPA_PERMITS
P [AND] POWERING_PERMITS
P [AND] RCBX_PERMITS
P [AND] RING_B1_PERMIT
  L [OR] BUNCH_LENGTH_RAMP_B1
  L [OR] RING_B1_SBF_MASKED
  L [AND] RING_PHYS_B1
    L [OR] ORBIT_PHYS_B1
      | BEAM_MODE_NOT_STABLE
      L [AND] ORBIT_READING_PHYS_HV_B1
        | ORBIT_READING_PHYS_H_B1
        | ORBIT_READING_PHYS_V_B1
  | TDE_PRESSURE_B1_PIEZO_TEMP_OK
P [AND] RING_B2_PERMIT
P [AND] RING_PERMIT
P [AND] RP_PERMITS
P [AND] SQUEEZE_FACTOR_BUILDER
P [AND] WARNING_PERMITS
```

```
Analysis (RING_B1_PERMIT.RING_PHYS_B1.ORBIT_PHYS_B1.ORBIT_READING_PHYS_HV_B1.ORBIT_READING_PHYS_V_B1)

Events            Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1
                  Event on LHC.USER.ALL:HX.BPNM-CT, counter=1

Channel state     UNMASKED [SYSTEM, Fri Apr 13 12:33:35 CEST 2018, As defined in configuration] UNLATCHED
Evaluation on     [1523970552000000000] Apr 17, 2018 3:09:12 PM on [LHC.USER.ALL:HX.BPNM-CT]
Evaluation result true
Cause             OK
Exception
Values used       UpdateEvent on LHC.USER.ALL:HX.BPNM-CT   tgmTelegram == <Click for value>   tgmTelegram == <Click for value>
                  SISREF_ORBPHYS == <Click for value>
                  Result:
                  received from SIS.BPM.REF/SisBpmRefPpm -> Header[acqSt
                  toleranceH (double[]:1076) -> 0.5, 0.5, 0.5, 0.5, 0.5,
                  toleranceV (double[]:1076) -> 0.5, 0.5, 0.5, 0.5, 0.5,
                  names (String[]:1076) -> BPM.33L1.B1, BPM.32L1.B1, BPM
                  dumpCount (int:1) -> 15
                  enabledV (boolean[]:1076) -> true, true, true, true, t
                  signature (String:1) -> 7899d1c39a977cc85242311b1ed995
```

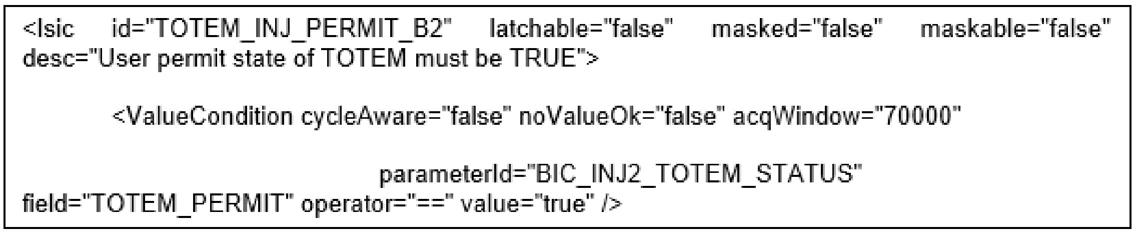
Figure 16.3: Permit tree for LHC ring 1

At the node level, specific behaviour can be configured. For example, a node can be masked to always return true. A node can also be latched, meaning that once it has been evaluated to false it will stay false, even if the next evaluation is true. It will stay in this state until an operator manually unlatches it using the SIS GUI or API.

The system is also fault-tolerant by configuring a counter, which allows a node to be evaluated to false, only once it has been consecutively evaluated to false a certain number of times; thus protecting the system from transient false-positives.

Similarly to other high-level acquisition and processing components, SIS evaluates the trees when triggered. It relies on JAPC Monitoring (see section 14.3) to acquire the device properties and generate triggering events.

Permits can be defined in two ways. XML for the simple expressions or a Groovy Domain Specific Language (DSL) for more powerful constructs. The XML approach is preferred by the users due to its simplicity, especially since one can reference Java classes in the XML in order to extend the logic with Java code. Figure 16.4 gives an example ISIC definition in XML. Since the definition of nodes can be quite repetitive for large machines, it is possible to use Velocity as a templating language to generate the XML configuration. The majority of the permits are written using a combination of XML and Velocity. The DSL option is seen by most users as a complex approach, which is harder to work with, and has little added value, as facilities for debugging and testing are not available.



```
<Isic    id="TOTEM_INJ_PERMIT_B2"    latchable="false"    masked="false"    maskable="false"
desc="User permit state of TOTEM must be TRUE">

        <ValueCondition cycleAware="false" noValueOk="false" acqWindow="70000"

                                parameterId="BIC_INJ2_TOTEM_STATUS"
field="TOTEM_PERMIT" operator="==" value="true" />
```

Figure 16.4: ISIC definition in XML

Performance is a key parameter in all interlock systems. The typical evaluation time for a permit is in the range of 200-300ms, for most of the use-cases, this is satisfactory. However, we have permits for the PS stray-field compensation in LEIR and PSB which have to acquire, evaluate and act within 1 basic period i.e. 1.2 seconds (see chapter 4).

Started in 2005, SIS is now a mature component and is almost feature complete. The operators are usually able to write their own logic independently. On the other hand, improvements to the permit documentation (history of evaluations, result traceability, list of actions taken) is required. Today, most of the information used to understand permit results are stored at the GUI level, thus limiting the availability and depth of permit history. This will be relocated to the server-side. In the future, it is foreseen to replace the XML, Groovy and Velocity configuration by a more unified alternative, such as Kotlin. This will provide much-needed features such as debugging, interoperability with Java code and IDE integration etc. The JMON Acquisition layer will also be replaced with a new acquisition layer based on the UCAP initiative (see section 15.4). The new layer should offer better tools to visualise, supervise and test the input signals of an application.

17. User Interfaces and Tools

At the top of the Control System stack are the Graphical User Interfaces[11] (GUI). These GUIs are used by a diverse user community, from accelerator operators and physicists, to equipment experts. The BE-CO group provides several customised graphical toolkits and frameworks to build specific accelerator applications. In addition, the group supplies a few applications to interact generically with the accelerator equipment.

17.1 Graphical Frameworks and Components

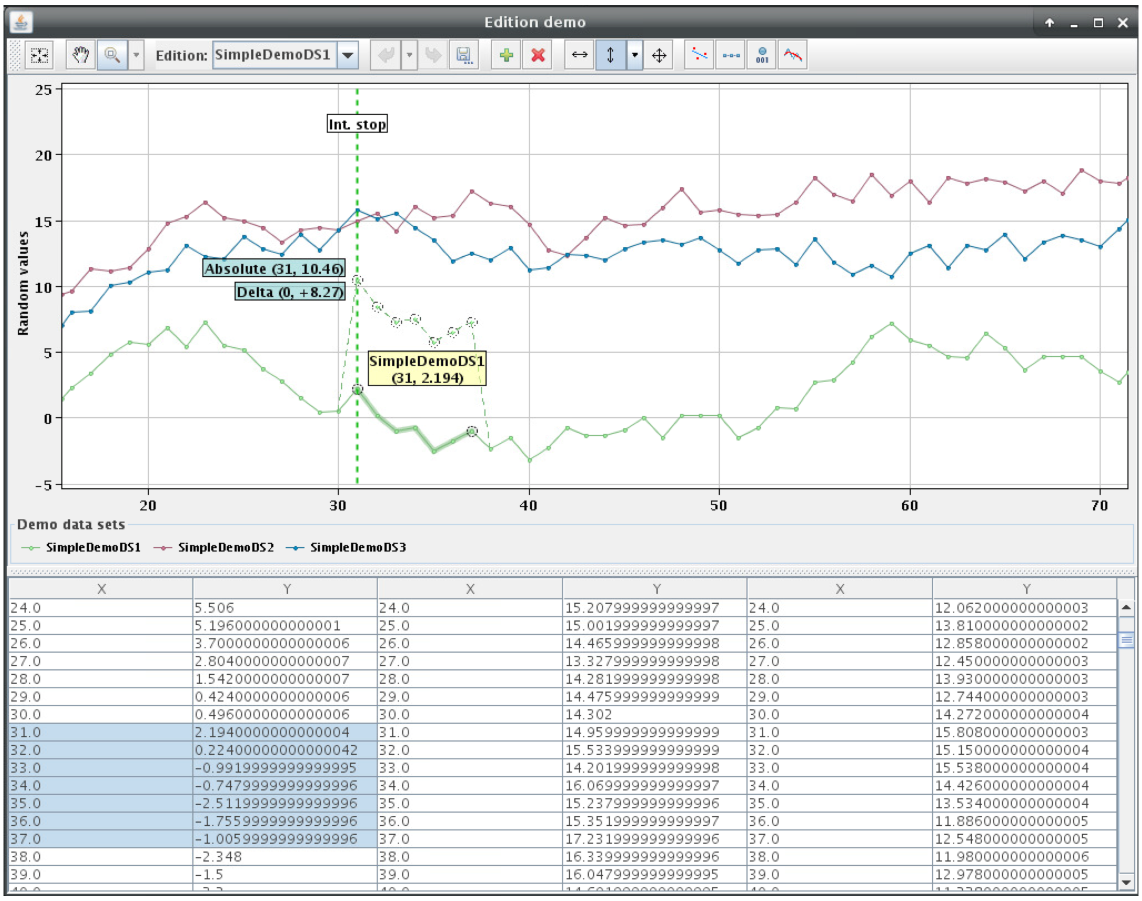
BE-CO has developed several frameworks and toolkits of components in order to streamline and standardise the development of Graphical User Interfaces. Some frameworks are non-accelerator specific, such as JDataViewer and Accsoft Commons Web (ACW), whereas others are aware of core concepts of the accelerator Control System, such as timing and the device property model (see chapters 3 and 4). For the construction of the LHC, the technology of choice for developing GUIs was Java and its main toolkit, Swing. However, since 2018, it is clear that the evolution of Java is moving away from graphical applications, even for JavaFX, which is much more recent than Swing. In parallel, web technologies are now the de-facto standard in industry for GUIs and the Python language has gained enormous popularity. Therefore, the future investment in the group are towards these two technologies and the first initiatives are described in sections 17.1.4 and 17.1.5.

17.1.1 JDataViewer

The JDataViewer covers the charting needs of the Control System. It was initially developed because there were no free, open-source alternatives available with the required feature set [37]. JDataViewer has the performance to display multiple graphs efficiently. In addition,

[11]The term Human Machine Interface (HMI) is also commonly used to refer to this layer of the Control System.

it supports direct, graphical data editing for modifying functions. Figure 17.1 depicts the point edition functionality of the JDataViewer. The first version was developed in the early 2000s, but has since undergone heavy refactoring in order to obtain the required performance. The current version of this mature component is written using a combination of Swing and Java2D. A few years ago, a JavaFX version was also developed to cover CERN's special needs not fully fulfilled by the JavaFX charting package [36].



X	Y	X	Y	X	Y
24.0	5.506	24.0	15.207999999999997	24.0	12.062000000000003
25.0	5.196000000000001	25.0	15.001999999999997	25.0	13.810000000000002
26.0	3.7000000000000006	26.0	14.465999999999998	26.0	12.858000000000002
27.0	2.8040000000000007	27.0	13.327999999999998	27.0	12.450000000000003
28.0	1.5420000000000007	28.0	14.281999999999998	28.0	13.930000000000003
29.0	0.4240000000000006	29.0	14.475999999999999	29.0	12.744000000000003
30.0	0.4960000000000006	30.0	14.302	30.0	14.272000000000004
31.0	2.1940000000000004	31.0	14.959999999999999	31.0	15.808000000000003
32.0	0.22400000000000042	32.0	15.533999999999999	32.0	15.150000000000004
33.0	-0.9919999999999995	33.0	14.201999999999998	33.0	15.538000000000004
34.0	-0.7479999999999996	34.0	16.069999999999997	34.0	14.426000000000004
35.0	-2.5119999999999996	35.0	15.237999999999996	35.0	13.534000000000004
36.0	-1.7559999999999996	36.0	15.351999999999997	36.0	11.886000000000005
37.0	-1.0059999999999996	37.0	17.231999999999996	37.0	12.548000000000005
38.0	-2.348	38.0	16.339999999999996	38.0	11.980000000000006
39.0	-1.5	39.0	16.047999999999995	39.0	12.978000000000005

Figure 17.1: JDataViewer point edition

17.1.2 AscBeans and Frame

The AscBeans make up a graphical toolkit of reusable components, which are aware of Control System concepts. Thanks to this built-in knowledge, AscBeans are data-driven and can work out of the box, just by specifying a JAPC parameter name and an accelerator context name. This is achieved by retrieving all of the required information from data services (mainly the InCA server), via descriptors. JAPC provides three types of descriptors; the device, parameter, and value descriptors. A descriptor is a map of configuration data and each descriptor provides information at different levels. For example, the parameter descriptor provides parameter information such as the PPM-ness and the value descriptor focuses on information about the value itself, such as its type and format pattern.

AscBeans allow operators and application developers to create GUIs to interact with the accelerator devices in an easy way, without having to manage common concerns.

For the communication, the AscBeans handle the subscription and data synchronisation, ensuring that all pieces of information belong to the same accelerator cycle. For the graphical aspects, the AscBeans render parameters according to their descriptors, selecting an appropriate component depending on the value type. Figure 17.2 depicts an example of an enumerated parameter as a combo box on the left, and a continuous numeric value as a wheel switch on the right.

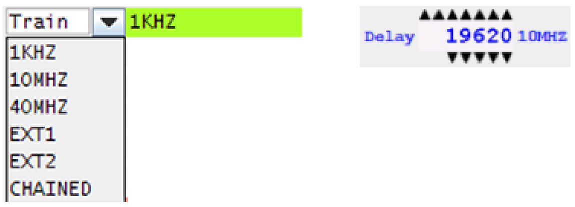
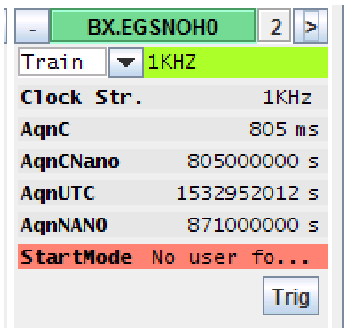


Figure 17.2: ASC Bean for enumerated and continuous numeric values

In addition, the AscBeans use the parameter's status coming from the InCA Acquisition Core (see section 15.1.2) to set the background colour, as standardised by Operations. Figure 17.3 shows an example of a healthy parameter with a green background and a parameter in an error state in red.



Clock Str.	1KHz
AqnC	805 ms
AqnCNano	805000000 s
AqnUTC	1532952012 s
AqnNANO	871000000 s
StartMode	No user fo...

Figure 17.3: Parameter status displayed in ASC Beans

All of the AscBeans provide a contextual menu giving access to several tools, such as the PPM comparator and the trim history viewer. Furthermore, there is also a large set of directly accessible diagnostic tools. As visible on figure 17.4, the user has access to diagnostic tools across all of the different layers, such as the JAPC/RDA diagnostic tool (aka JAPC toolbox) for the client-server communication, the FESA navigator for the front-end layer and the TGM tools for the timing system.

In addition to the AscBeans, the Frame (accsoft-gui-frame) provides the foundations for an application. By default, all GUIs based on the frame, are provided with a menu bar and a toolbar with generic tools, such as accelerator context selection, as well as a console to display error messages and diagnostic traces.

The current implementations of AscBeans and the Frame are in Java Swing, but the concept and early versions date from the 90s and were implemented in X-motif. The future evolution of the product will depend on the group strategy chosen for the graphical user interface technologies (JavaFX, Qt, Web). In addition, a few improvements are also

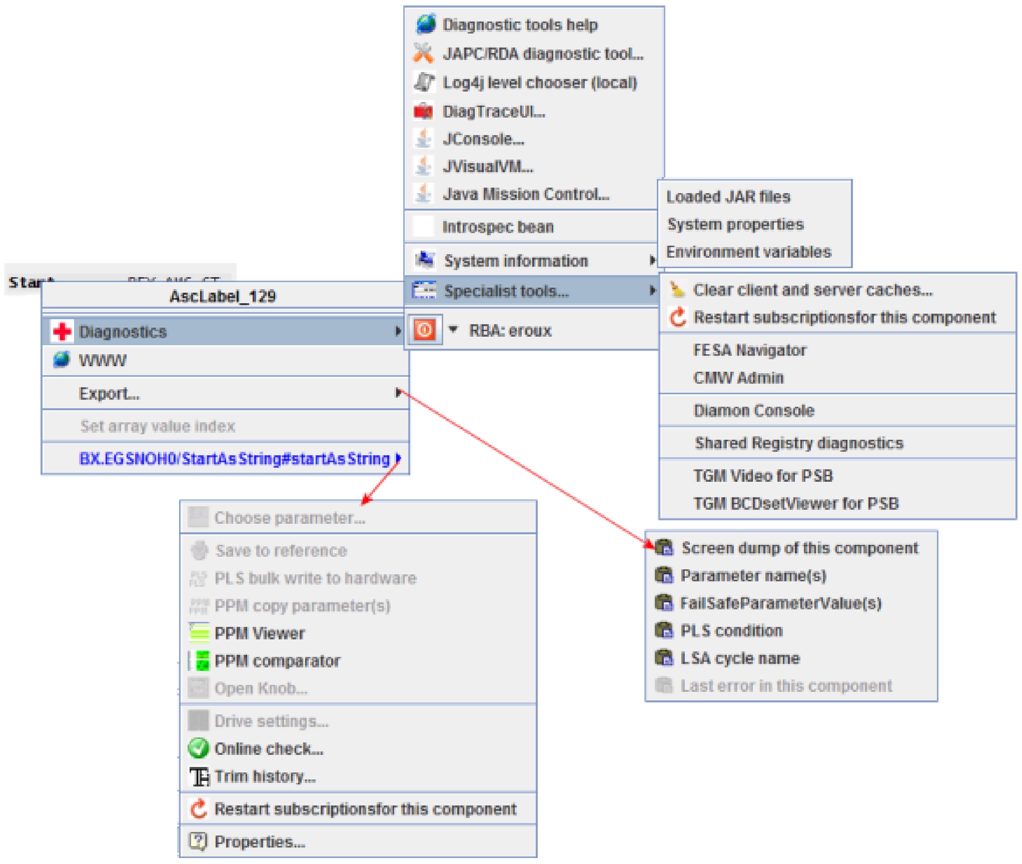


Figure 17.4: Contextual menu for an AscLabel

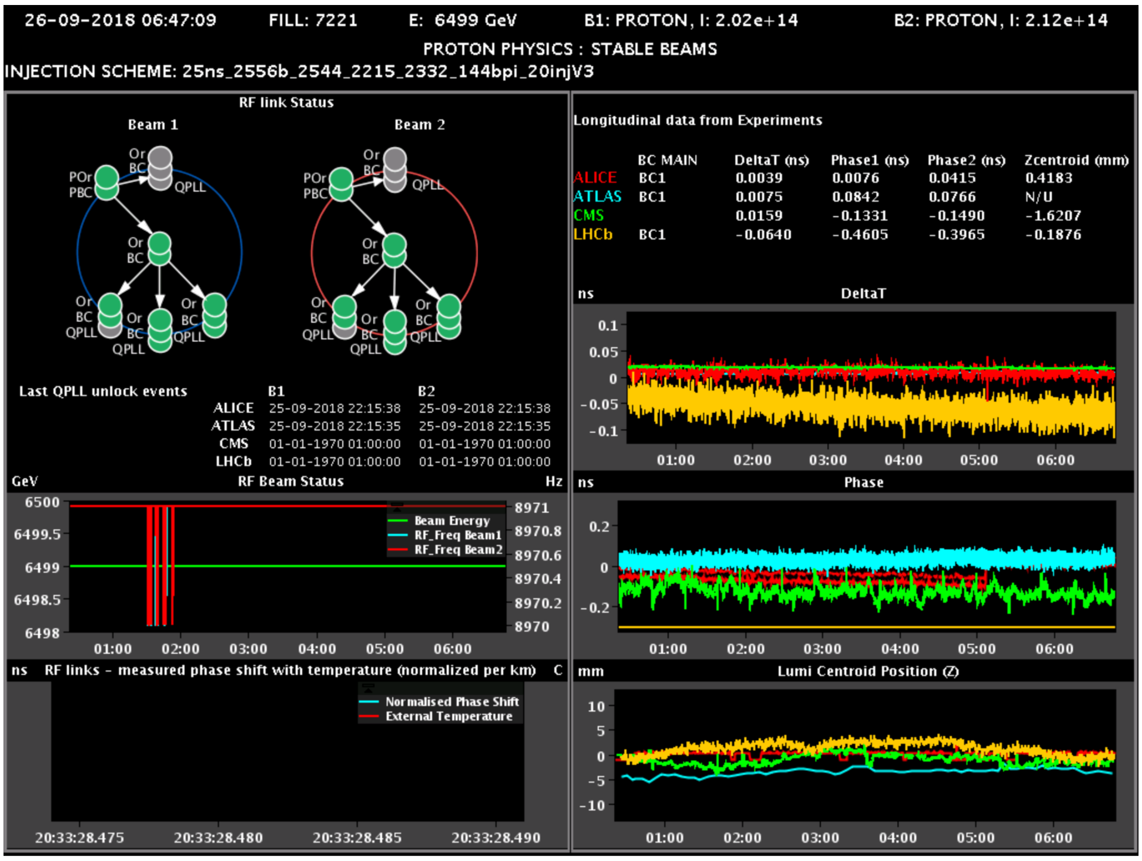
requested, such as more flexibility to select the source of data (InCA vs JavaScript Object Notation (JSON) file) at runtime.

17.1.3 FDF

There are many cases when interaction with the Graphical User Interface is not required. Typically, these are read-only visualisations of acquired data, displayed on the web or on big television screens in control rooms. As there are many such cases, it was decided to develop a tool, the Fixed Display Framework (FDF), to rationalise their development.

At the low-level, FDF relies on JAPC Monitoring to acquire and synchronise the data. Every Fixed Display is implemented as a JAPC Monitoring module (see section 14.3.1). An XML file is used to describe the data sources and layout of the graphical components, which are based on Swing, ASC Beans and JDataViewer. Figure 17.5 shows a fixed display for LHC RF timing.

For the web distribution and the video streaming, the fixed display instances run on servers. The web server reads a screenshot of the fixed display (png file) that is periodically created on the local disk, every 1-5 seconds. For the video streaming, depending on the latency requirements, two solutions are used. For low-latency cases, a graphics card produces the video output and a hardware MPEG-4 encoder encodes the video stream, before it is distributed on the Ethernet network. When high-performance is not needed, a video card is



26-09-2018 06:47:09 FILL: 7221 E: 6499 GeV B1: PROTON, I: 2.02e+14 B2: PROTON, I: 2.12e+14

PROTON PHYSICS : STABLE BEAMS

INJECTION SCHEME: 25ns_2556b_2544_2215_2332_144bpi_20injV3

Longitudinal data from Experiments

	BC MAIN	DeltaT (ns)	Phase1 (ns)	Phase2 (ns)	Zcentroid (mm)
ALICE	BC1	0.0039	0.0076	0.0415	0.4183
ATLAS	BC1	0.0075	0.0842	0.0766	N/U
CMS		0.0159	-0.1331	-0.1490	-1.6207
LHCb	BC1	-0.0640	-0.4605	-0.3965	-0.1876

Last QPLL unlock events

	B1	B2
ALICE	25-09-2018 22:15:38	25-09-2018 22:15:38
ATLAS	25-09-2018 22:15:35	25-09-2018 22:15:35
CMS	01-01-1970 01:00:00	01-01-1970 01:00:00
LHCb	01-01-1970 01:00:00	01-01-1970 01:00:00

Figure 17.5: Fixed Display for LHC RF timing

not used and instead the screen is rendered in memory and encoded by software.

Initially, the framework was intended to be employed by end-users to develop their screens. However, the number of developments and their frequency is not sufficient for the users to master the framework and its various technologies (JMON, Swing, XML, etc.). Therefore, BE-CO performs the development based on a specification (graphics and text), source of data, and expected refresh rate.

Whilst the framework is very stable, several drawbacks need to be addressed. Firstly, in order to allow users to work alone, a simplification of the acquisition and processing part will be done by replacing JAPC Monitoring with UCAP (see section 15.5).

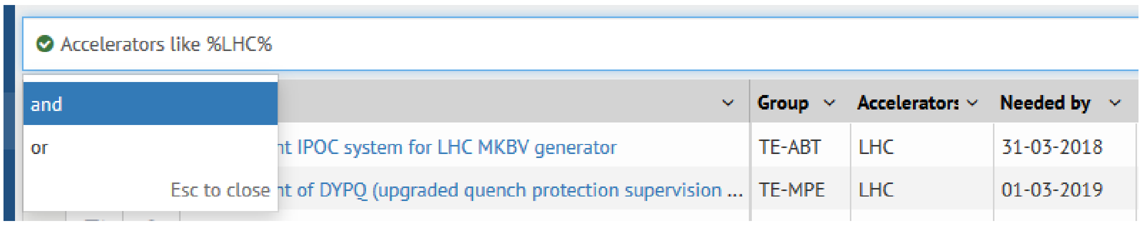
With the future of Swing being unclear, a replacement toolkit will have to be chosen. Potential candidates are JavaFX, or Qt. Maintainability should also be improved. Currently, all required parameters are hardcoded in the XML file, so when there are migrations, or devices are renamed, this can easily break a Fixed Display. It is hard to proactively detect these events in a fixed display. Finally, more efficient and cost effective ways of streaming video and web-distribution should be found.

17.1.4 ACW

Many applications need to be easily accessible from any computer without prerequisites. For such cases, web applications were developed, as all users are accustomed to these types of applications. Furthermore, this allowed the group to introduce technologies that are popular in the outside world, and hence facilitate recruitment. Previous BE-CO web applications relied on expensive proprietary products not popular in industry, e.g. Oracle Apex, and another motivating factor was to move towards more widely used open-source technologies.

One of the challenges of web development is the large choice of frameworks and technologies available and their rapid evolution (new frameworks, library versions, forks, etc.). Therefore, the aim of the Accelerator software Commons Web (ACW) initiative was to develop and evolve a standard approach to web development in BE-CO, upon which custom web applications can be based. The idea is that by having a common approach overall, we will keep development and maintenance costs down, whilst increasing the flexibility for people to work on one web application or another. Moreover, it is important that the applications provided to our user community have a common look and feel wherever possible.

ACW offers a template for the full stack: from client, to server, to database access. In particular, ACW provides a number of shared aspects including the configuration and management of common application dependencies, essential features such as security (Single-Sign-On and RBAC (see section 13.2)), and application instrumentation. On the client-side, ACW includes some common web application components (e.g. time range selections, tree navigation, data grids, searching, data input forms, validations, etc.), as well as application menus and navigation management. Figure 17.6 shows one of the ACW components, the DSL search.


Accelerators like %LHC%

and
or
Esc to close

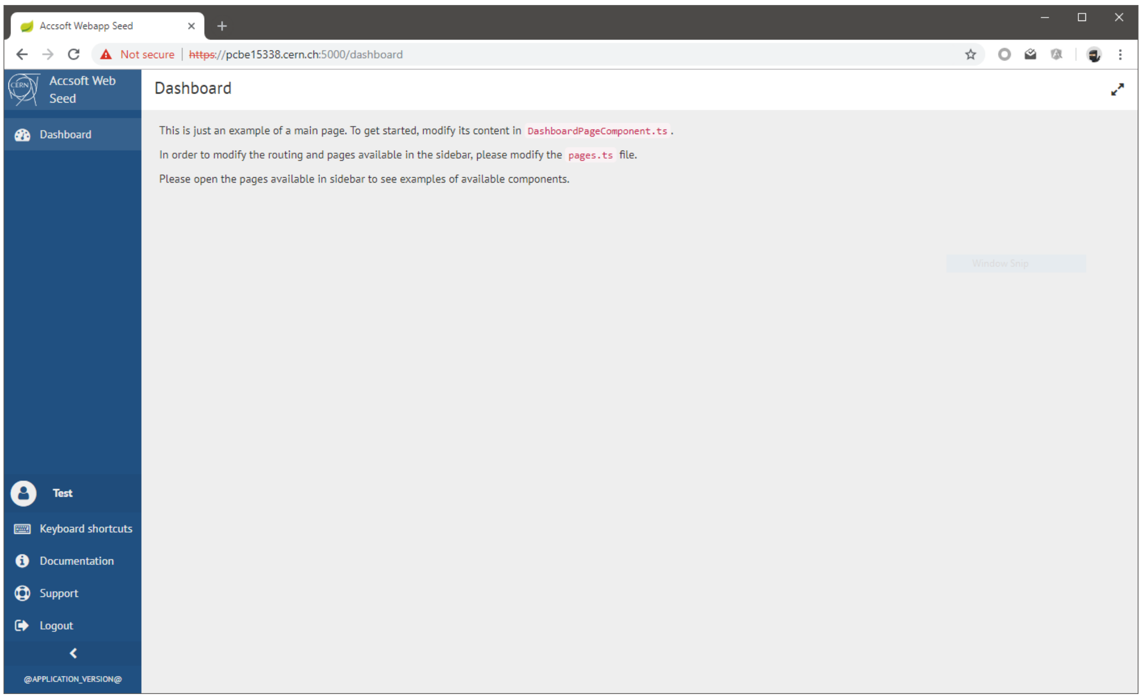
	Group	Accelerator:	Needed by
ıt IPOC system for LHC MKBV generator	TE-ABT	LHC	31-03-2018
ıt of DYPQ (upgraded quench protection supervision ...	TE-MPE	LHC	01-03-2019

Figure 17.6: ACW DSL search component with auto-completion and validation support

The ACW server-side software is written in Java, in order to profit from the group's expertise, and uses Spring and Spring Boot. The client-side is written in TypeScript and Sassy Cascading Style Sheets (SCSS), an extension of Cascading Style Sheets (CSS), and uses the Angular JS framework and libraries such as Bootstrap and Font Awesome. Additionally, tools such as Webpack, Node.js Package Manager (NPM) and Gradle are integrated into ACW to package applications, manage dependencies, and support the development and software building processes. In order to start a new project based on the ACW framework, a user would check out the so-called 'Seed' project[12] and fork it into their own project. In a very short period of time, a client-server application with minimal content, such as the one shown in figure 17.7 can be created.

The main challenge is to identify developments in individual ACW-based applications that should be consolidated into ACW. It is also a challenge to coordinate ACW developments

[12] `https://gitlab.cern.ch/accsoft-commons-web`



This is just an example of a main page. To get started, modify its content in `DashboardPageComponent.ts`.

In order to modify the routing and pages available in the sidebar, please modify the `pages.ts` file.

Please open the pages available in sidebar to see examples of available components.

Figure 17.7: ACW seed application

in a way that ensures evolution of the framework in a coherent way, without negatively impacting existing applications. In addition, extra time has to be dedicated to maintaining and evolving ACW on top of the development of individual ACW-based applications. Nevertheless, there is a clear return on investment.

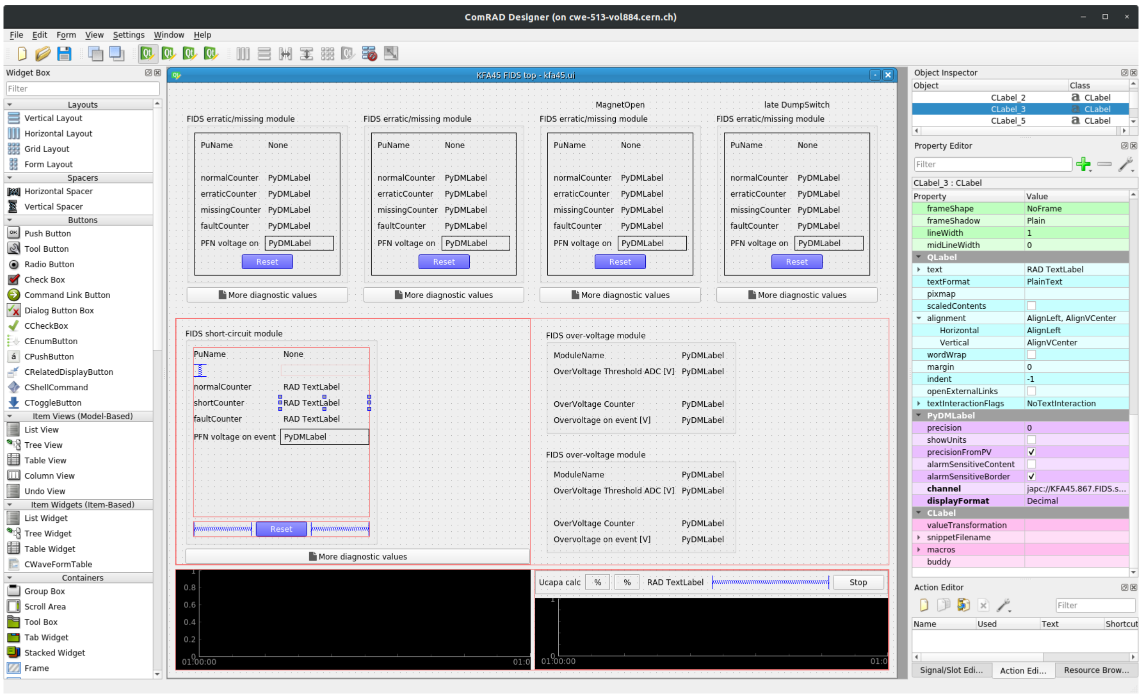
In 2019, the short-term foreseen evolution includes upgrading the core underlying frameworks (e.g. AngularJS to Angular, Bootstrap 3 to 4 and Spring 4 to 5) and adapting the ACW software accordingly.

17.1.5 COMRAD

The Python language has gained popularity over recent years and is a suitable alternative to the declining Java technologies on the client-side. Indeed, since 2018, it is clear that the evolution of Java is moving away from graphical applications, for both JavaFX and Swing. For these reasons, the group has decided to support Python in the Control System. At the same time, we have taken the opportunity to review the different Java graphical components in order to assess their relevance after 15 years of development. In addition, there were strong requests from the user communities to provide a tool with which to develop simple applications without writing code, or at least as little as possible. Our solution to these requests is COntrols Multi-purpose Rapid Application Development (COMRAD).

The first objective of the new tool is to provide a drag-and-drop development environment using already available widgets, connected to the Control System via CMW. Later, a better integration with the Control System will be provided with services such as device discovery, as well as new components inspired by AscBeans. Finally, we aim to fulfil the frequently expressed need of being able to evolve expert prototypes into operational applications [33]. Therefore, it should be possible for COMRAD prototype applications to be converted into regular PyQt applications, thus unleashing the full power of PyQt.

To save time and resources, COMRAD is based on a pre-existing solution. From the particle accelerator community, PyDM from Stanford Linear Accelerator Center (SLAC) [47] and Taurus from the TANGO collaboration [44] were evaluated against the requirements. PyDM was selected for several technical reasons, including having a newer, smaller code-base written in Python 3 and being PyQt 5 ready. On top of PyDM, a CERN-specific layer provides integration with the Control System through PyJAPC, in-house widgets, as well as the pre-configuration of the tool and its widgets. BE-CO is collaborating with SLAC by contributing to the stability of the core PyDM product with bug-fixes. CERN-developed features could also be upstreamed, if relevant to the wider community. For performing the drag-and-drop design of the GUIs, we rely on the de-facto tool, Qt Designer, as shown in figure 17.8.



Object	Class
CLabel_2	CLabel
CLabel_3	CLabel
CLabel_5	CLabel

CLabel_3 : CLabel

Property	Value
frameShape	NoFrame
frameShadow	Plain
lineWidth	1
midLineWidth	0
QLabel	
text	RAD TextLabel
textFormat	PlainText
pixmap	
scaledContents	☐
alignment	AlignLeft, AlignVCenter
Horizontal	AlignLeft
Vertical	AlignVCenter
wordWrap	☐
margin	0
indent	-1
openExternalLinks	☐
textInteractionFlags	NoTextInteraction
PyDMLabel	
precision	0
showUnits	☐
precisionFromPV	✓
alarmSensitiveContent	☐
alarmSensitiveBorder	✓
channel	japc://KFA45.867.FIDS.s...
displayFormat	Decimal
CLabel	
valueTransformation	
snippetFilename	
macros	
buddy	

Name	Used	Text	Shortcut

Figure 17.8: COMRAD development environment

17.2 Generic Applications

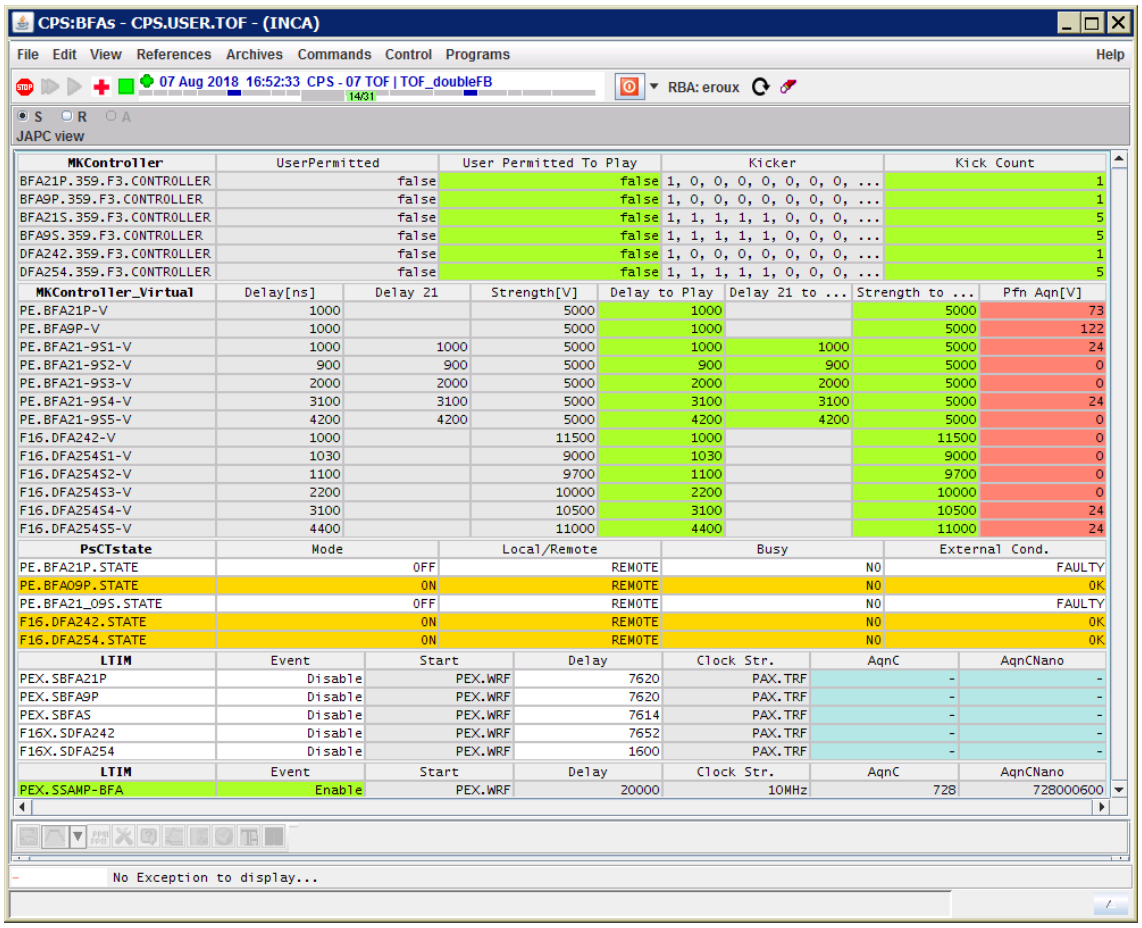
Accelerators are complex to operate and require many high-level applications. However, thanks to the standardisation done at the Control System-level (device-property model, timing selectors, etc.), the BE-CO group is able to provide generic applications that fulfil many control and acquisition use cases. In addition, the group has to provide a means of launching applications and managing them, taking into account the multiplexed nature of the accelerators.

17.2.1 Working Sets, Knobs and Function Editor

For the smaller accelerators, the operators prefer to interact with the control devices displayed as structured lists and tables, a WorkingSet. This allows them to look at part of their machine in one glance and to open control applications easily. Unfortunately, this approach cannot be applied to large accelerators such as the LHC, due to the number of devices.

A WorkingSet is made of a number of independent tables. Each table (device group) contains a number of devices, for which key parameters are displayed. The list of key parameters and their position in the table are configured in the InCA database.

The WorkingSets are based on AscBeans, and as explained in section 17.1.2, the AscBeans set their background colour depending on the status of the parameter. This allows operators to immediately identify potential issues. Figure 17.9 depicts a WorkingSet with five device groups and their parameters. Settings diverging from the reference value and acquisitions deviating from their control value are displayed in orange (warning) and red (error) respectively. Inactive parameters are displayed in white, and green indicates a healthy device or parameter.



MKController	UserPermitted	User Permitted To Play	Kicker	Kick Count
BFA21P.359.F3.CONTROLLER	false	false	1, 0, 0, 0, 0, 0, 0, 0, ...	1
BFA9P.359.F3.CONTROLLER	false	false	1, 0, 0, 0, 0, 0, 0, 0, ...	1
BFA21S.359.F3.CONTROLLER	false	false	1, 1, 1, 1, 1, 0, 0, 0, ...	5
BFA9S.359.F3.CONTROLLER	false	false	1, 1, 1, 1, 1, 0, 0, 0, ...	5
DFA242.359.F3.CONTROLLER	false	false	1, 0, 0, 0, 0, 0, 0, 0, ...	1
DFA254.359.F3.CONTROLLER	false	false	1, 1, 1, 1, 1, 0, 0, 0, ...	5

MKController_Virtual	Delay[ns]	Delay 21	Strength[V]	Delay to Play	Delay 21 to ...	Strength to ...	Pfn Aqn[V]
PE.BFA21P-V	1000		5000	1000		5000	73
PE.BFA9P-V	1000		5000	1000		5000	122
PE.BFA21-9S1-V	1000	1000	5000	1000	1000	5000	24
PE.BFA21-9S2-V	900	900	5000	900	900	5000	0
PE.BFA21-9S3-V	2000	2000	5000	2000	2000	5000	0
PE.BFA21-9S4-V	3100	3100	5000	3100	3100	5000	24
PE.BFA21-9S5-V	4200	4200	5000	4200	4200	5000	0
F16.DFA242-V	1000		11500	1000		11500	0
F16.DFA254S1-V	1030		9000	1030		9000	0
F16.DFA254S2-V	1100		9700	1100		9700	0
F16.DFA254S3-V	2200		10000	2200		10000	0
F16.DFA254S4-V	3100		10500	3100		10500	24
F16.DFA254S5-V	4400		11000	4400		11000	24

PsCTstate	Mode	Local/Remote	Busy	External Cond.
PE.BFA21P.STATE	OFF	REMOTE	NO	FAULTY
PE.BFA09P.STATE	ON	REMOTE	NO	OK
PE.BFA21_09S.STATE	OFF	REMOTE	NO	FAULTY
F16.DFA242.STATE	ON	REMOTE	NO	OK
F16.DFA254.STATE	ON	REMOTE	NO	OK

LTIM	Event	Start	Delay	Clock Str.	AqnC	AqnCNano
PEX.SBFA21P	Disable	PEX.WRF	7620	PAX.TRF	-	-
PEX.SBFA9P	Disable	PEX.WRF	7620	PAX.TRF	-	-
PEX.SBFAS	Disable	PEX.WRF	7614	PAX.TRF	-	-
F16X.SDFA242	Disable	PEX.WRF	7652	PAX.TRF	-	-
F16X.SDFA254	Disable	PEX.WRF	1600	PAX.TRF	-	-

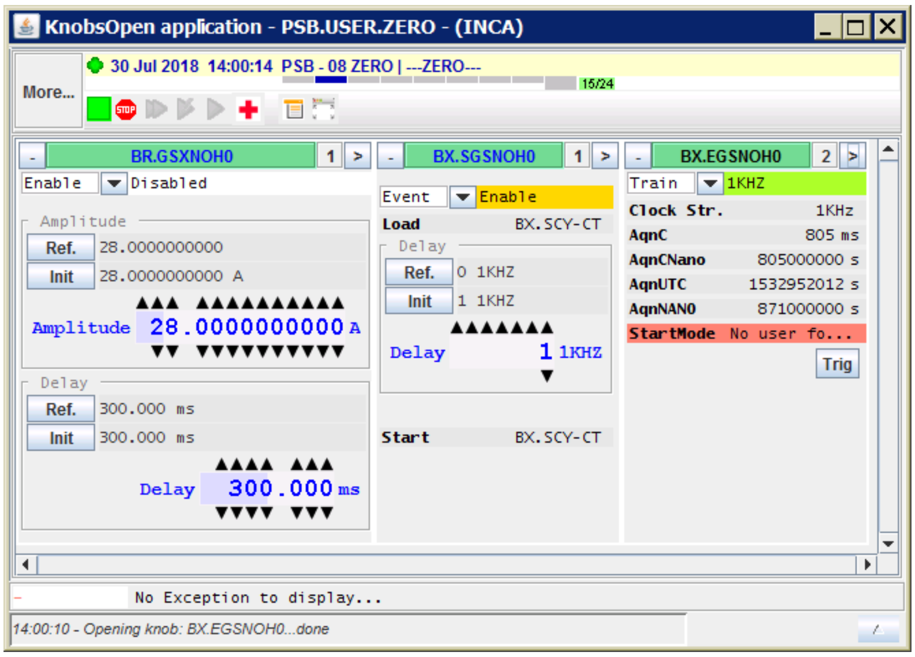
LTIM	Event	Start	Delay	Clock Str.	AqnC	AqnCNano
PEX.SSAMP-BFA	Enable	PEX.WRF	20000	10MHz	728	728000600

```
No Exception to display...
```

Figure 17.9: WorkingSet showing PS beam extraction devices for the TOF timing user

While the WorkingSets are mainly used for monitoring, they still facilitate a few a control actions and bulk operations, such as setting all of the devices to a given value, or switching them off. In addition, from the WorkingSets, a set of applications for editing the values can be launched, already pre-configured with the selected device and timing context. The list of available applications depends on the class of the selected device. Two of these applications, the knobs and the Function Editor, are widely used to control scalar values and functions of time respectively.

A knob is opened by double-clicking on a device in a WorkingSet. Figure 17.10 shows three knobs for different device classes, the last two knobs control local timings, the first of which displays the main page and the other the second page.


KnobsOpen application - PSB.USER.ZERO - (INCA)

30 Jul 2018 14:00:14 PSB - 08 ZERO | ---ZERO--- 15/24

More...

BR.GSXNOH0	1
Enable	Disabled
Amplitude	
Ref.	28.0000000000
Init	28.0000000000 A
Amplitude	28.0000000000 A
Delay	
Ref.	300.000 ms
Init	300.000 ms
Delay	300.000 ms

BX.SGSNOH0	1
Event	Enable
Load	BX.SCY-CT
Delay	
Ref.	0 1KHZ
Init	1 1KHZ
Delay	1 1KHZ
Start	BX.SCY-CT

BX.EGSNOH0	2
Train	1KHZ
Clock Str.	1KHz
AqnC	805 ms
AqnCNano	805000000 s
AqnUTC	1532952012 s
AqnNANO	871000000 s
StartMode	No user fo...
	Trig

No Exception to display...

14:00:10 - Opening knob: BX.EGSNOH0...done

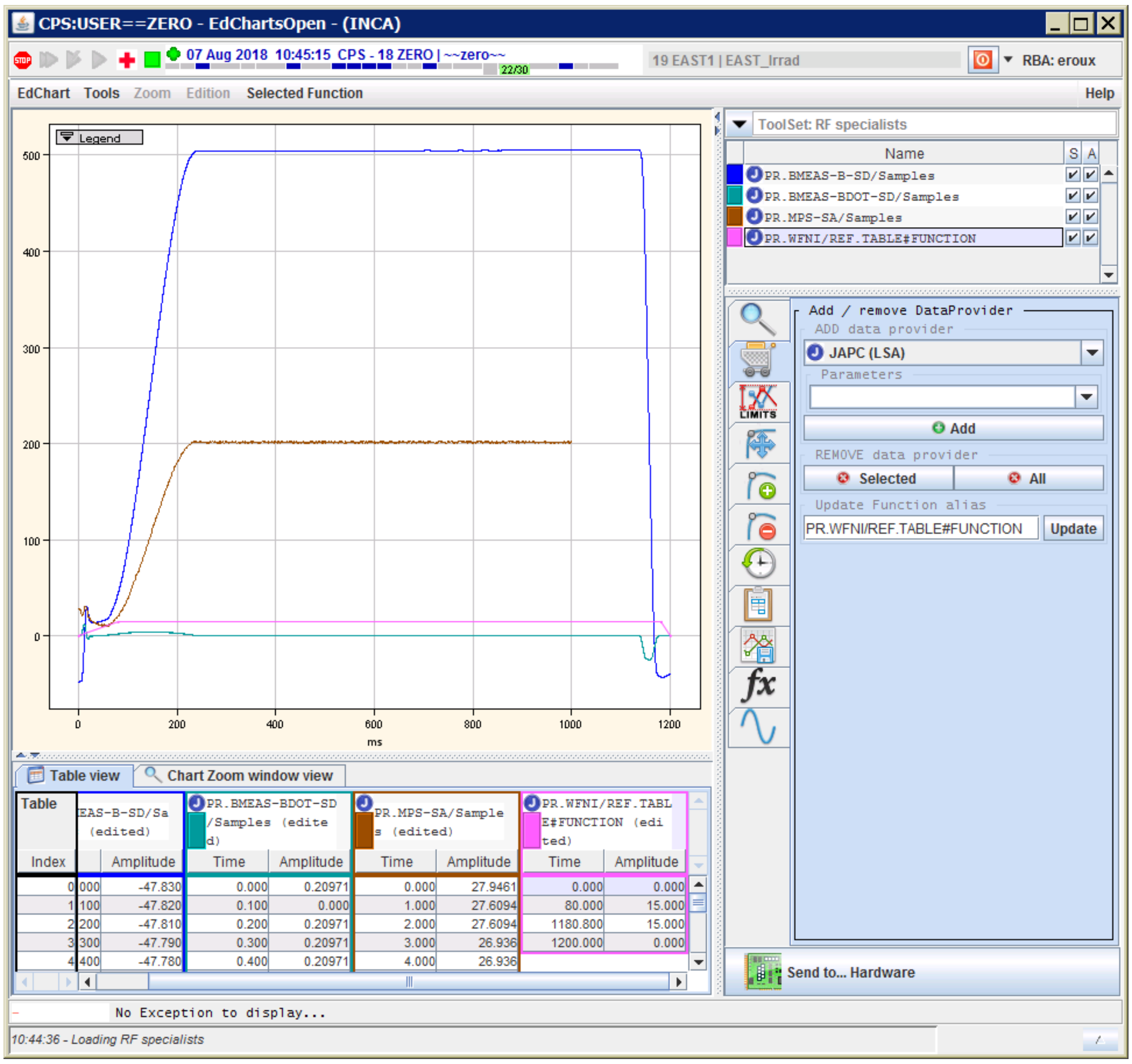
Figure 17.10: Three knobs to control a function generator and two local timings

Similarly to the WorkingSets, the knobs configure themselves automatically, according to the device class and the pre-configured layout, stored in the InCA database. As the knobs are also implemented using AscBeans, the rendering of the individual parameters is set in function of the value type (e.g. continuous numeric values are rendered as a wheel switch, whereas an enumerated value uses a combo box (see figure 17.2).

For the parameters representing functions over time, there is another generic application called the Function Editor. It can be launched from the WorkingSets whenever the selected device has one or more parameters of type function or function list[13]. The Function Editor allows basic function editing (adding points, removing points), plus more advanced features such as inserting mathematical sub-functions to a curve. To facilitate the work, the Function Editor can display several functions from different devices at the same time. Figure 17.11 depicts a Function Editor with data coming from four different devices.

As already mentioned, these generic applications are based on AscBeans and are AscBeans themselves. Therefore all of the tooling and diagnostics available from the contextual

[13]A function list is an ordered list of functions, where the continuity between adjacent functions is guaranteed.



	Name	S	A
🟦	PR.BMEAS-B-SD/Samples	✓	✓
🟩	PR.BMEAS-BDOT-SD/Samples	✓	✓
🟫	PR.MPS-SA/Samples	✓	✓
🟪	PR.WFNI/REF.TABLE#FUNCTION	✓	✓

Table	EAS-B-SD/Sa (edited)		PR.BMEAS-BDOT-SD /Samples (edited)		PR.MPS-SA/Samples (edited)		PR.WFNI/REF.TABLE#FUNCTION (edited)	
Index		Amplitude	Time	Amplitude	Time	Amplitude	Time	Amplitude
0	000	-47.830	0.000	0.20971	0.000	27.9461	0.000	0.000
1	100	-47.820	0.100	0.000	1.000	27.6094	80.000	15.000
2	200	-47.810	0.200	0.20971	2.000	27.6094	1180.800	15.000
3	300	-47.790	0.300	0.20971	3.000	26.936	1200.000	0.000
4	400	-47.780	0.400	0.20971	4.000	26.936		

```
No Exception to display...
```

10:44:36 - Loading RF specialists

Figure 17.11: Function Editor, open with three acquired parameters and the programmed function of a PS power converter

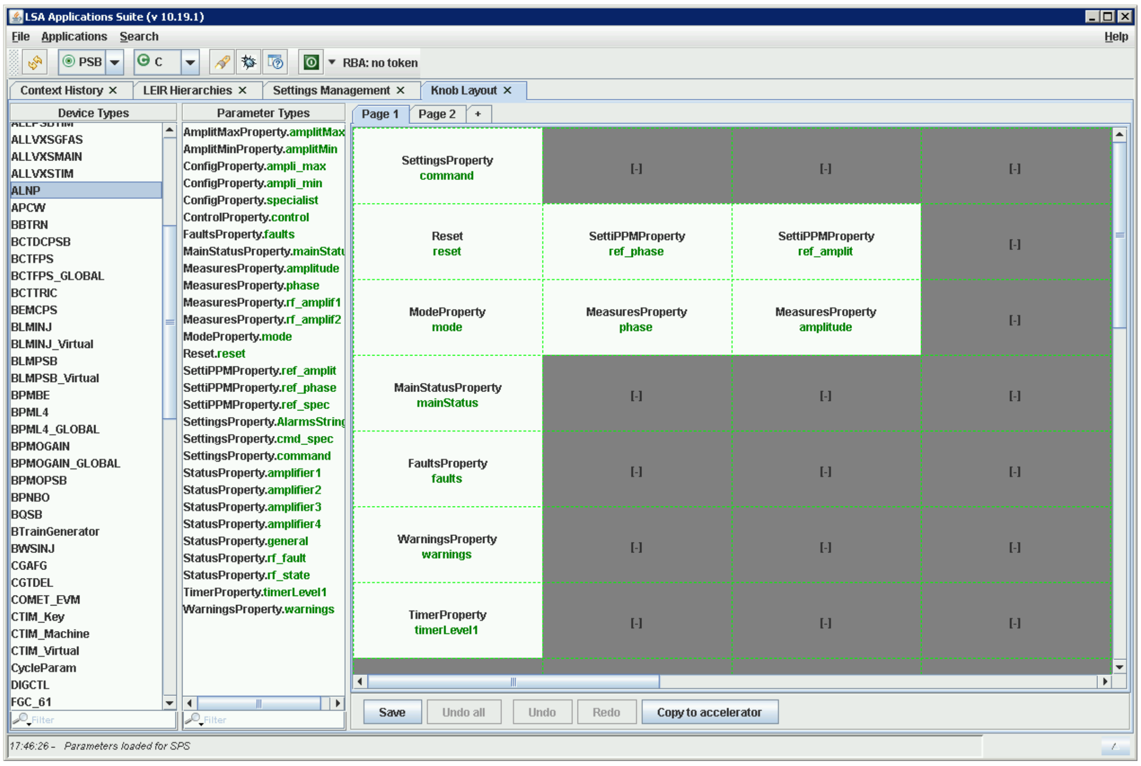
menu are also accessible and an application developer can embed WorkingSets, knobs and Function Editors in any application.

Operating the PS complex accelerators using mainly WorkingSets and knobs has been possible since the 90s. The first implementation was in C++ and X-motif, but was re-implemented in Java Swing in the early 2000s. Since the graphical design was completed, the Control System's infrastructure has evolved and more data types, such as 2D arrays, are now supported. One of the upcoming challenges is to display these new types in a user-friendly way. In the recent years, low-level FESA classes have become more complex, and it is becoming difficult to display all of the available parameters in a 2D table.

17.2.2 LSA Application Suite

Another generic application is the LSA Application Suite, a single GUI that gathers together several tools, which used to be independent. The suite provides the means to trim, copy, compare and regenerate parameter settings and clone and map cycles. In addition, it can be used to configure knob and Working Set layouts and device groups that are used in other applications. Figure 15.1 depicts the setting Management tab in the LSA Application Suite.

Figure 17.12 shows the knob layout configuration tab. The reason to group these different panels into a single application is to provide a more integrated user experience. With the LSA Application Suite, the user does not have to launch many separate applications, and configure them individually with the same pieces of information such as accelerator, context type, cycles etc. each time. Nevertheless, since each accelerator has its specificities in terms of operations, the suite itself must be highly configurable in order to satisfy OP requirements.



Device Types
ALLVXSGFAS
ALLVXSMAIN
ALLVXSTIM
ALNP
APCW
BBTRN
BCTDCPSB
BCTFPS
BCTFPS_GLOBAL
BCTTRIC
BEMCPS
BLMINJ
BLMINJ_Virtual
BLMPSB
BLMPSB_Virtual
BPMBE
BPML4
BPML4_GLOBAL
BPMOGAIN
BPMOGAIN_GLOBAL
BPMOPSB
BPNBO
BQSB
BTrainGenerator
BWSINJ
CGAFG
CGTDEL
COMET_EVM
CTIM_Key
CTIM_Machine
CTIM_Virtual
CycleParam
DIGCTL
FGC_61

Parameter Types
AmplitMaxProperty.amplitMax
AmplitMinProperty.amplitMin
ConfigProperty.ampli_max
ConfigProperty.ampli_min
ConfigProperty.specialist
ControlProperty.control
FaultsProperty.faults
MainStatusProperty.mainStatu
MeasuresProperty.amplitude
MeasuresProperty.phase
MeasuresProperty.rf_amplif1
MeasuresProperty.rf_amplif2
ModeProperty.mode
Reset.reset
SettiPPMProperty.ref_amplit
SettiPPMProperty.ref_phase
SettiPPMProperty.ref_spec
SettingsProperty.AlarmsStrin
SettingsProperty.cmd_spec
SettingsProperty.command
StatusProperty.amplifier1
StatusProperty.amplifier2
StatusProperty.amplifier3
StatusProperty.amplifier4
StatusProperty.general
StatusProperty.rf_fault
StatusProperty.rf_state
TimerProperty.timerLevel1
WarningsProperty.warnings

SettingsProperty command	[-]	[-]	[-]
Reset reset	SettiPPMProperty ref_phase	SettiPPMProperty ref_amplit	[-]
ModeProperty mode	MeasuresProperty phase	MeasuresProperty amplitude	[-]
MainStatusProperty mainStatus	[-]	[-]	[-]
FaultsProperty faults	[-]	[-]	[-]
WarningsProperty warnings	[-]	[-]	[-]
TimerProperty timerLevel1	[-]	[-]	[-]

Figure 17.12: LSA Application Suite with the knob layout configuration panel

More applications should be integrated into the LSA Application Suite, such as the context management application and the parameter configuration tool. However, since the app suite is written in Swing and, in 2019, the future of the latter is unclear, development is currently on hold.

17.2.3 Common Console Manager

The Common Console Manager (CCM) is the operator's entry point to all of the controls applications. The CCM offers a user-specific, multi-level menu bar from which the operators launch their GUIs. The configuration of the menus is based on an Operational Configuration (OPConfig) stored in the CCDB (see section 20.1). When logging on using a recognised service account, the appropriate configuration is loaded, with all of the menus and applications preconfigured for that team. Figure 17.13 shows the CCM menu bar for the LHCOP operational configuration.

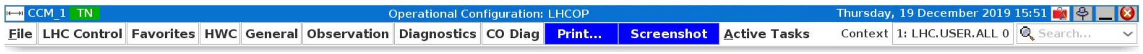


Figure 17.13: CCM menu bar for LHCOP

The CCM is also context-aware, and, for applications supporting timing contexts, the CCM will manage the application windows according to the context selection. For example, if the first context is CPS.USER.SFTPRO, and the operator switches to another context (e.g. CPS.USER.LHC), the CCM will minimise all windows of all applications using the first context, and restore the windows for all applications open in the second context.

The CCM communicates with the applications that it launches using the Shared Registry (SHREG), a small Java server storing structured key values, giving the timing context, selected device names, etc.

The first implementation of the CCM was also in X-motif, but a renovation project was launched in the early 2000s in order to migrate it to Java Swing. Even though the current implementation is in Java, since the CCM runs on several platforms (Windows, Linux), it requires native code to interact with each platform's window manager. In addition, the association between a window appearing on a screen and the owning application is based on the window title. These two mechanisms make the CCM application management quite fragile and could be improved.

18. High-Level Development Tools

A large amount of high-level software development is done within the group, facilitated by a suite of high-level development tools. For the IDE, the policy is relatively lenient, with a several different products being used, including Eclipse and IntelliJ. On the other hand, the development workflow, i.e. how to build, release, deploy and version code, should be as homogeneous as possible, within a given technology. Indeed, the different development teams should not have to consider these aspects for every new project and duplication of effort should be avoided whenever possible. The BE-CO group provides solutions implementing industry's best practices and relying on third-party products.

18.1 Best Practices

To ensure maintainability and high availability of the Control System we strive to follow software development best practices from industry. The process elements such as version control and continuous integration are facilitated using third-party tools, which are customised to CERN-specific needs.

18.1.1 Version Control

Over the years, the Controls group has used several Version Control Systems (VCSs). The first tool, Revision Control System (RCS), versioned files individually and locally, until tools such as Razor, and then SVN, with centralised repositories became the norm. In 2018, CO embraced the IT department's strategy to move away from SVN towards Git, which corresponds to the general trend in industry. The IT department has chosen a product called GitLab to provide the Git infrastructure. Nevertheless, Git is conceptually quite different to SVN, with the main challenge being moving from a mono-repository to many individual repositories. For the group, being able to access the entire code base is important for maintenance, and therefore the migration to Git requires a review of the tooling. Indeed, when planning backward incompatible changes in APIs , a study of their impact must be carried out in order to understand the consequences on the users.

18.1.2 Continuous Integration

Modern software development teams should use Continuous Integration (CI) with unit and system tests, in order to minimise the risk of regression bugs. BE-CO has used Atlassian Bamboo for several years to provide a CI infrastructure. The tool allows each team to define one or several plans to test the software they produce. Additionally, since the controls stack relies on several components, the tool provides a plan dependency mechanism, whereby modifications in a plan triggers the execution of other plans, thus validating the complete chain.

Following the Long Shutdown 1 (LS1), the group went a step further in its CI by setting up the Controls Testbed (CTB), in order to validate the next version of the controls stack (FESA, CMW, timing, etc.). Code changes are validated against several FESA classes, to ensure that there are no adverse side-effects in other components. Therefore, a seemingly innocent modification containing a bug, which passes the unit tests, would be caught by the integration tests performed in the CTB.

The latest evolution of the VCS, with new products such as GitLab, offers new CI solutions. However, the industry trend is towards open-source products, such as Jenkins. In 2019, the group is evaluating the two possibilities, in order to rationalise the tools used for CI.

18.2 Build Process

Different build processes have been implemented, depending on the technology, and specific tools are provided to facilitate their adoption. For Java, the entire process, from building through to release and deployment, is handled by Common Build Next Generation (CBNG), while for other technologies, such as Python, only standard third-party tools are used.

18.2.1 CBNG

CBNG is a CERN-made tool that helps to perform most of the Java development operations, such as dependency management, integration with VCSs, and building. Historically, there was no solution available on the market that fitted our needs, and with more than 500 Java products, it was important to provide an easy-to-use tool for both full-time developers and operators. Dependency management alone is a very complex subject and justifies the creation of a tool such as CBNG that fulfils the CERN-specific requirements. In addition, a centrally managed tool allows global operations such as bug fixes to be performed.

From a user's point of view, interaction with CBNG happens through a file called `product.xml` and a set of commands, which can be invoked either through an IDE, such as Eclipse, or the command line (e.g. `bob build`), in the same spirit as one would use `make compile` in C++. In the `product.xml` file, the user describes their product (name, versions etc.), the dependencies of the product and some information to allow the generation of a launch script. Thanks to the dependency information, the tool can fetch the libraries in a recursive manner, implying that it will fetch the dependencies of the dependencies until none remain. Figure 18.1 shows a simplified example of a product depending on three libraries, with the libraries themselves having dependencies. Note how quickly the dependency management problem becomes complex, as the product already uses two different versions of lib4. This kind of situation has to be handled properly by the CBNG tool.

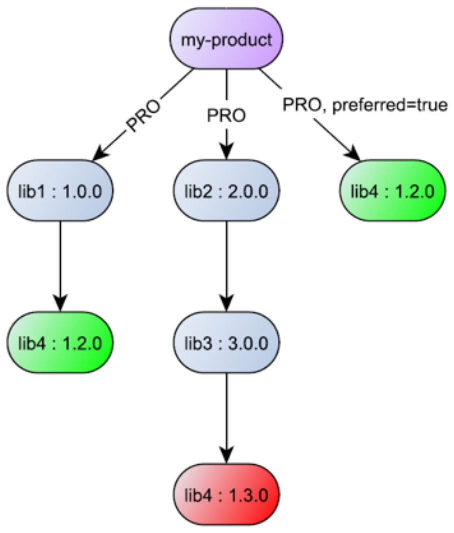


Figure 18.1: A simple dependency graph with a conflict in transitive dependencies

The BE-CO group has more than a thousand Java components, which are highly inter-dependent. This is shown on the dependency graph in figure 18.3.

Two other core features of CBNG are building and releasing the product. When building, the tool will compile the product code, its unit test, and execute the unit tests.

Once the software component is ready to be deployed in operation, the last stage of the development is called the release. The release process is very similar to the normal build, except that traceability and reproducibility must be ensured. Indeed, since the piece of software will be deployed in production, anybody must be able to rebuild it from the source code, without any specific dependencies on a particular developer or development environment. Furthermore, in case of problems, it is important to be able to identify the exact version of the source code which caused the bug. For Java developments, CBNG integrates a release process, customised to CERN-specific needs. Figure 18.2 depicts a simplified version of the release process.

The reproducibility of the release is ensured thanks to tagging the source code with the version number and the use of a release server, which isolates the build from any specific development environment and relies only on the tagged sources, without any local dependencies. The end result is stored in the so-called Artifactory.

The deployment phase consists of taking released software from the Artifactory and transferring it to the computer on which it will be executed. In 2019, all of the processes are run bare-metal, directly on the computers. However, studies are ongoing to evaluate the possibility of using containers and clustering technologies. The execution of the processes are performed by an in-house product called wreboot. The process' health is monitored by COSMOS, as explained in chapter 19.

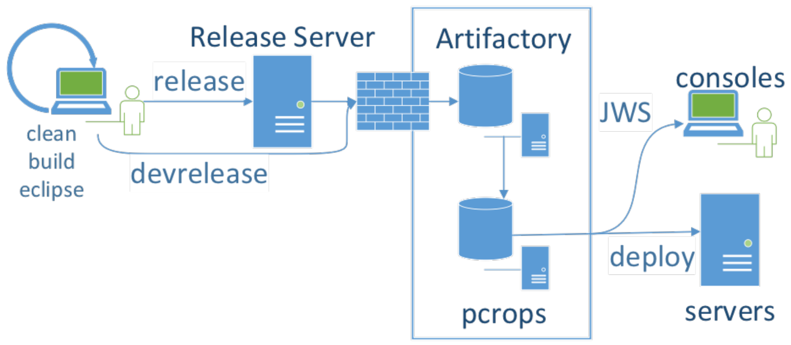


Figure 18.2: Simplified release process

In 2019, many third-party build tools are available on the market. That means, for newcomers, CBNG is seen as a CERN-specific non-standard approach. In the future CBNG will evolve towards Gradle [26]. However, with its latest release (V3), CBNG has become much closer to pure Gradle and users of the latter can now profit from CBNG integration thanks to a plug-in.

18.2.2 Python Build Tools

In 2019, the Python development infrastructure is still embryonic. A Python Package Index (PyPI) repository, to store CERN-specific Python artefacts, is in place. It proxies the global repository, holding the software developed and shared by the Python community. A BE-CO Python distribution is also available on all development and operational computers. It is based on the LHC Computing Grid (LCG)'s Python 3.6 distribution, packaged by the Experimental Physics (EP) department. In addition, we use the virtual environment mechanism to select the Python interpreter and the libraries to be used in a given development.

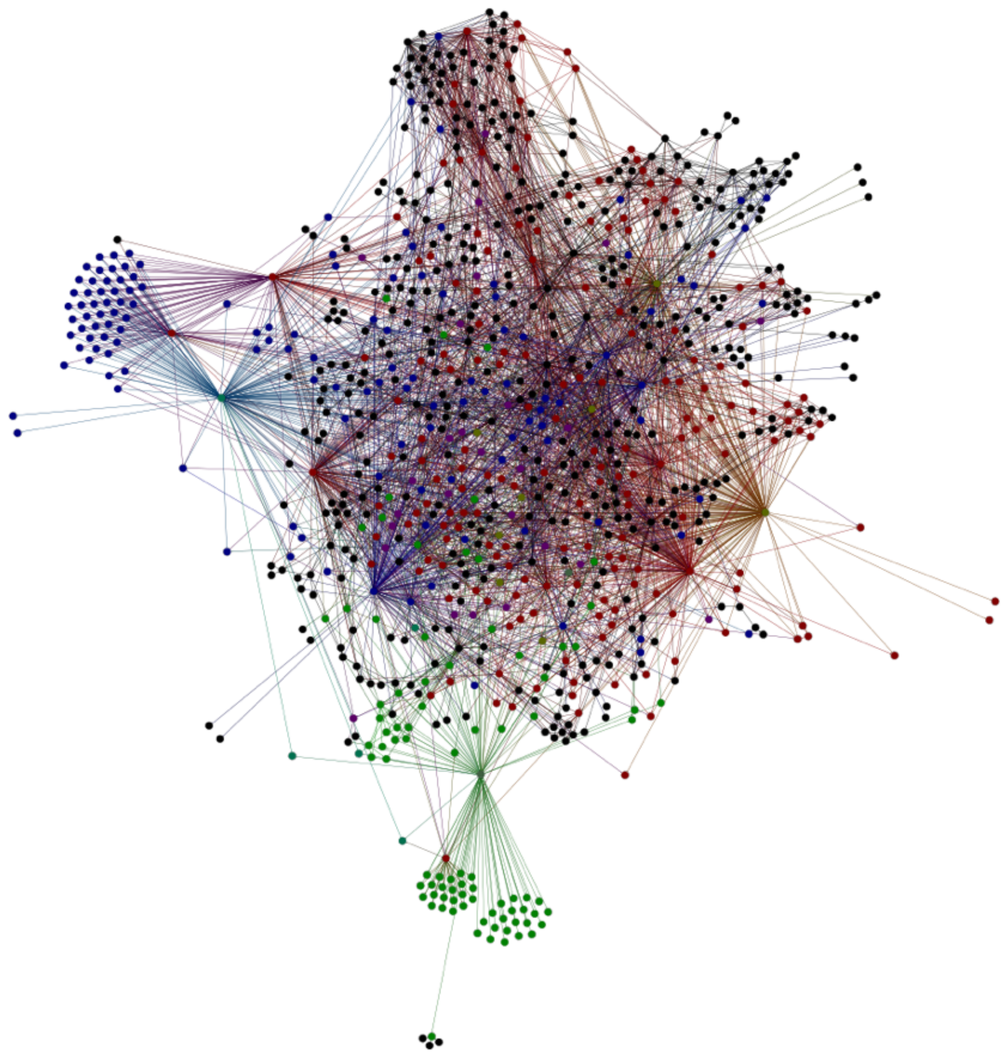


Figure 18.3: Dependency graph between BE-CO's Java components

VI Transversal Components

19. Monitoring, Testing and Diagnostics

19.1 COSMOS

As described in previous chapters, the Control System relies on many components and a substantial infrastructure, which needs to be available 24 hours a day, 7 days a week. Therefore, the infrastructure needs to be monitored in order to detect any failures and intervene as soon as possible. The Controls Open-Source Montoring System (COSMOS) project was launched in 2017, following a review of the different monitoring solutions. One of the main goals of COSMOS was to consolidate several systems such as Diagnostic and Monitoring System (DIAMON), LHC Era Monitoring (LEMON), etc. Contrary to previous solutions, the scope of COSMOS is well-defined and covers only the monitoring of the technical infrastructure, delegating to the GUI layer the integration of various services, such as process management, remote reset, etc.

At the heart of the COSMOS project, one finds an open-source product called Icinga2. Icinga2 is based on a distributed architecture in which the data is processed in the monitored nodes, rather than in the central server and introduces core concepts such as checks, health reports, and performance data. The health report, containing a status of the monitored component, is obtained via checks. Figure 19.1 shows the COSMOS architecture with its three main parts: sources, COSMOS core and visualisation.

COSMOS uses Collectd agents to gather the OS metrics from the hosts, related devices (disks, memory, etc.) and network. Collectd makes this information available over the network to the central server, where the data is stored in a time-series database, InfluxDB. In parallel, COSMOS uses the Icinga2 check mechanism to gather health reports, related performance data and thresholds from all hosts and services of the control infrastructure, using custom or dedicated protocols like IPMI or SNMP. It populates the central database used by the expert tools such as IcingaWeb, to establish, in real time, the complete

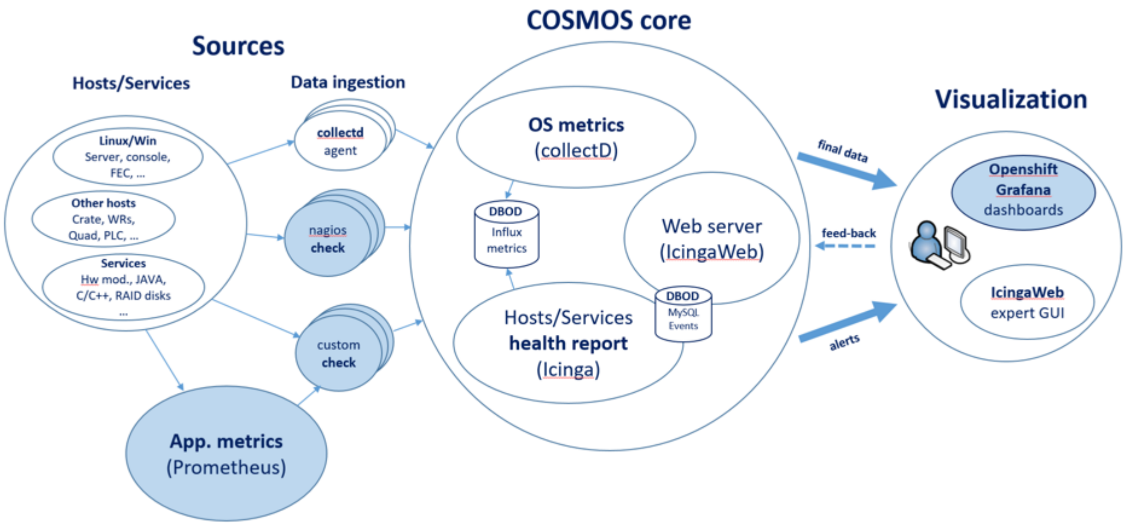
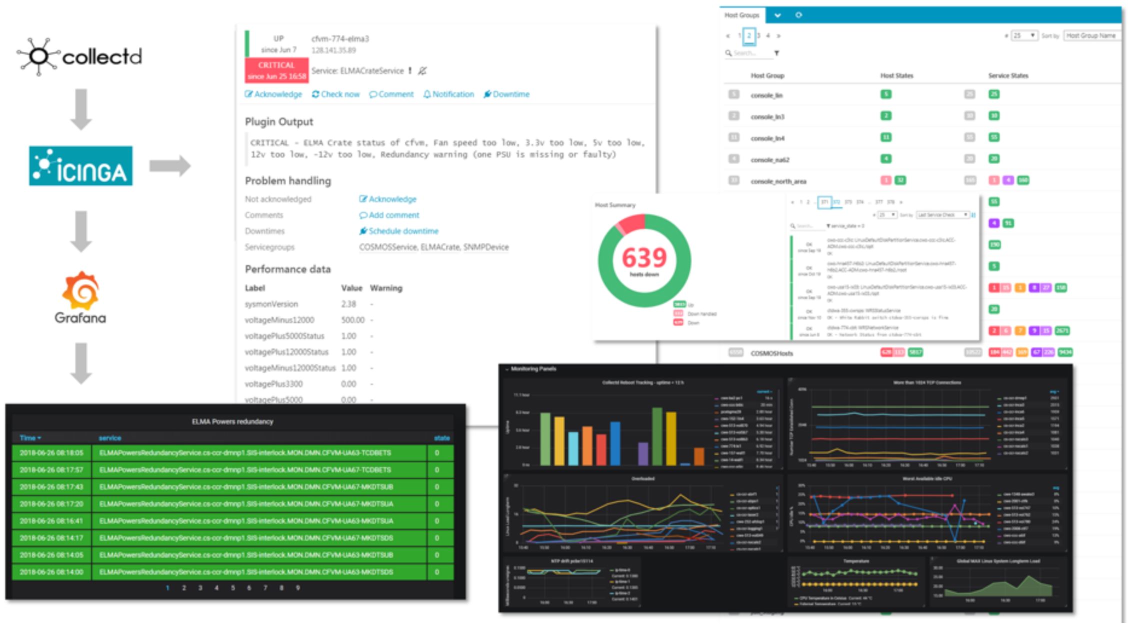


Figure 19.1: COSMOS architecture

diagnostic of each component, to configure user alerts, and to provide detailed statistics. Whenever it is not possible to compute the status of a service from a single Icinga2 check, or to detect trends, COSMOS uses Prometheus as an intermediate agent. Prometheus then generates the service status using its functional expression language. Finally, the data is visualised by the users, thanks to dedicated software, including Grafana or IcingaWeb, as shown in figure 19.2. Alerts are sent through the standard mechanisms of SMS and email.



```
CRITICAL - ELMA Crate status of cfvm, Fan speed too low, 3.3v too low, 5v too low,
12v too low, -12v too low, Redundancy warning (one PSU is missing or faulty)
```

Figure 19.2: COSMOS GUIs

Users of COSMOS can contribute to the system by developing their own Grafana dashboards and providing custom checks. Checks are pieces of code executed on the monitored nodes and can be as simple as a few lines of Bash, C or Python. As depicted in figure 19.3, checks can be active or passive. An active check is triggered by the central server in polling mode and pulls metrics from the nodes synchronously. A passive check occurs when a standalone agent pushes data to the central server asynchronously. In the case of systems where it is critical to start and stop processes at any moment, passive checks are recommended. For the same reason, it is suggested to use high-performing languages such as C or C++ for real-time systems. Indeed, for the BE-CO low-level frameworks, the interfacing with COSMOS is done with C++ Management Extension (CMX) agents, as described in the next chapter.

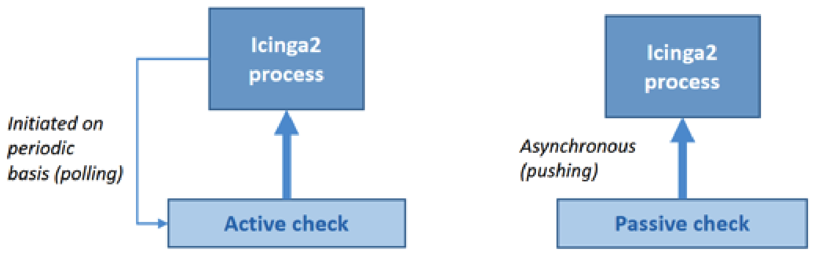


Figure 19.3: COSMOS checks

The first phase of the project focused on delivering the COSMOS hardware and software infrastructure in order to monitor most of the hosts deployed on the GPN and TN. The second phase concentrated on the consolidation and extension for advanced requirements such as volatile configuration, dependencies and relationships, diagnostics, and high-availability. In addition, it also covers the monitoring of applications such as Java services and C++ real-time processes. The operational DIAMON GUI will remain in place but will evolve to use COSMOS as a data source.

19.1.1 CMX

For RT systems, it is particularly important that monitoring does not interfere with control processes. Nevertheless, metrics and internal states of these processes must be gathered in order to detect potential issues as soon as possible. In 2013, CMX [25], was developed as a lightweight means to monitor our RT systems.

Inspired by JMX, CMX offers similar functionality, but with a simpler set of features targeted to our specific needs. CMX provides non-blocking, uni-directional communication between the monitored process and shared memory, however it only supports numbers and strings. The equivalent of the JMX MBean is called a component, and by default every process has one component to gather usual Linux process information such as Process IDentification number (PID), start-time, etc. Libraries and frameworks can contribute additional components, for example FESA automatically adds specific components, depending on the class design. In order to make the library as lightweight as possible and be available to low-level processes, the CMX APIs are provided in C and C++.

19.2 Low-level Test Tools

When developing or diagnosing a Front-End application, it is very useful to have test tools that can interface directly with the layer under investigation. We provide tools for the direct access of the three layers present in a Front-End: hardware registers, device drivers and FESA servers.

19.2.1 Encore/EDGE Test Tool and UAL

`encoreconsole` allows the debugging of a hardware component through a Command Line Interface (CLI) that leverages the functionalities of the generic Encore/EDGE library. In particular, the registering of I/Os, DMA transactions, waiting for interrupts and a whole sequencing facility. Furthermore, re-running of previous simulated data is possible using the hardware module register naming, taken from the hardware description (see chapter 10).

In addition, a separate tool that bypasses the device driver installation, is provided: the Unified Access Library (UAL) and its Python binding, PyUAL. While EDGE generates production-quality drivers and libraries, UAL is meant for quick hardware testing in non-production context.

UAL is a simple library that maps the register map of a hardware device into user space, abstracting the host bus (PCI or VME) into a simple API, also exposed through Python bindings. The goal is to ease the development of test/validation programs for hardware during the design and production phase, and to streamline the creation of diagnostic tools for non-production purposes only.

The interrelation of components and artefacts is depicted in figure 19.4.

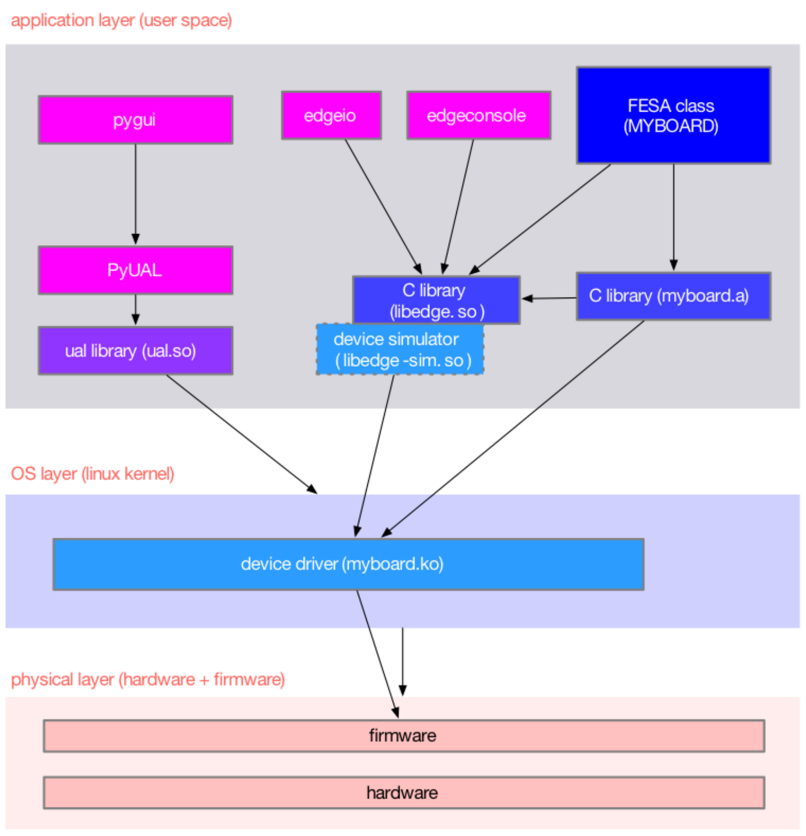


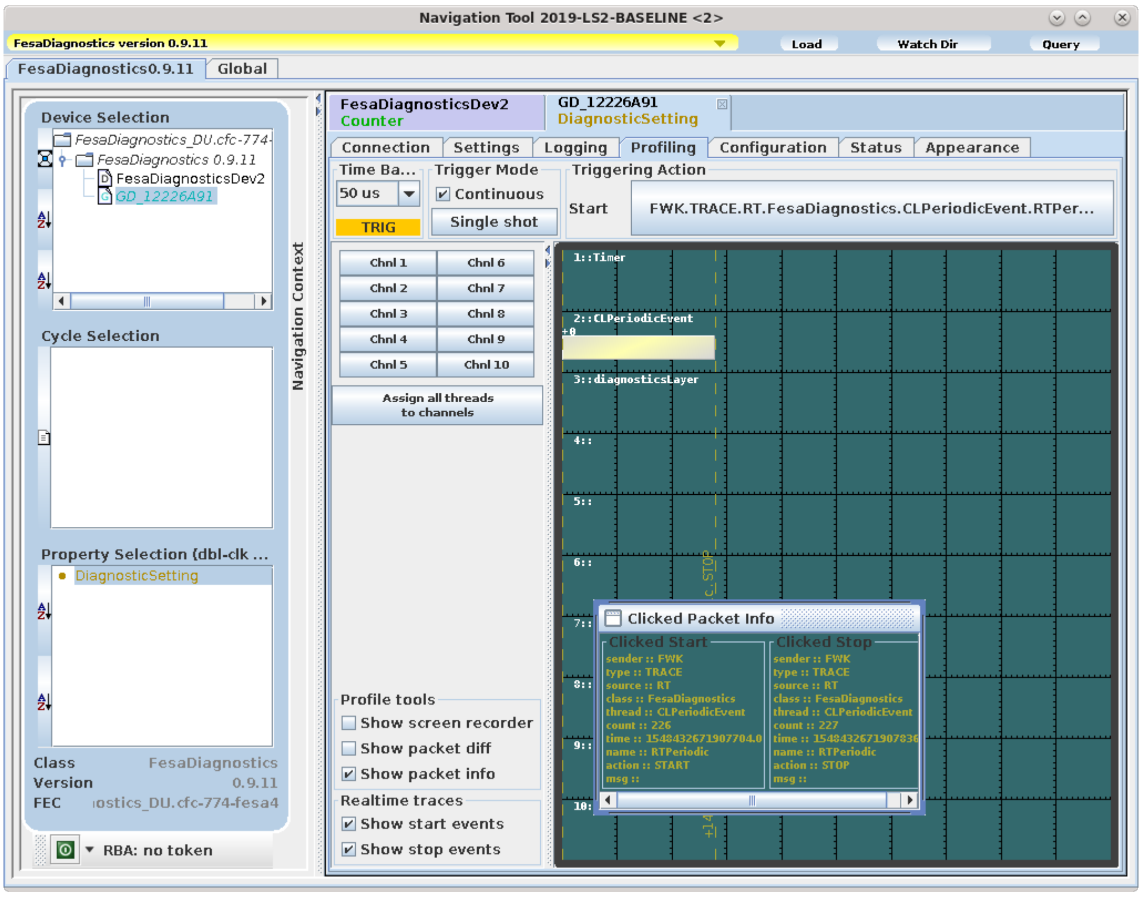
Figure 19.4: Components of EDGE and PyUAL for testing

19.2.2 FESA Navigator

Moving one level up in the controls stack, the FESA Navigator gives low-level, expert access to the FESA processes. The Navigator is the main toolbox used by the FESA class developer and provides raw access to all properties with a set of viewers supporting most of the data types, history, tracing and even real-time profiling, as shown in figure 19.5. In addition, users have the possibility to configure the panels in order to help visualise the properties of complex classes and save this layout for future use.

The Navigator can be used by experts while developing their classes, as well as during operation for low-level diagnostics. For that reason, it needs to obtain static information such as class design and device instances either from the FESA XML files or by connecting to the CCDB to retrieve operational data. Since the Navigator is a diagnostic tool, it should interface at the lowest possible level with the running FESA classes, without interference from the high-level layers. Therefore to retrieve runtime information it connects directly to the FESA servers through CMW-RDA3.

The FESA Navigator was developed at the same time as the first version of FESA in the early 2000s and since then has evolved significantly. It is based on the technologies available at that time, including Java Swing for the graphical part. In the future, it is planned to reimplement this tool, taking into account the two decades of experience and other available technologies such as PyQt.



Class FesaDiagnostics
Version 0.9.11
FEC ıostics_DU.cfc-774-fesa4

Clicked Packet Info

```
Clicked Start                         Clicked Stop
sender :: FWK                         sender :: FWK
type :: TRACE                         type :: TRACE
source :: RT                          source :: RT
class :: FesaDiagnostics              class :: FesaDiagnostics
thread :: CLPeriodicEvent             thread :: CLPeriodicEvent
count :: 226                          count :: 227
time :: 1548432671907704.0            time :: 1548432671907836
name :: RTPeriodic                    name :: RTPeriodic
action :: START                       action :: STOP
msg ::                                msg ::
```

Figure 19.5: FESA Navigator with profiling panel

19.3 Tracing

One of the most basic methods of diagnosing software is by adding tracing information. This approach is simple and efficient. Nevertheless, in a highly distributed system, such as the Control System, it can quickly become a challenge as the information is stored in many files from different origins on different computers. In order to ease the diagnostic work for everybody involved in software development (operations, equipment groups, etc.), BE-CO provides a tracing infrastructure with central storage and extraction tools. The tracing service gathers information from many data sources, mainly syslogs, CMW loggers including FESA and FGC servers, and Java applications.

The tracing architecture is comprised of many third-party, open-source components, as depicted in figure 19.6. At the centre of this architecture is a Kafka broker and an Elasticsearch cluster. Traces stored in files are injected into the service using Logstash, before being sent to the Kafka broker, while traces from CMW loggers are sent directly. A set of processors implementing a pull paradigm, take data from the broker, apply filters, and enhance the data before sending it to Elasticsearch. Kibana and Grafana are used to extract and post-process the traces.

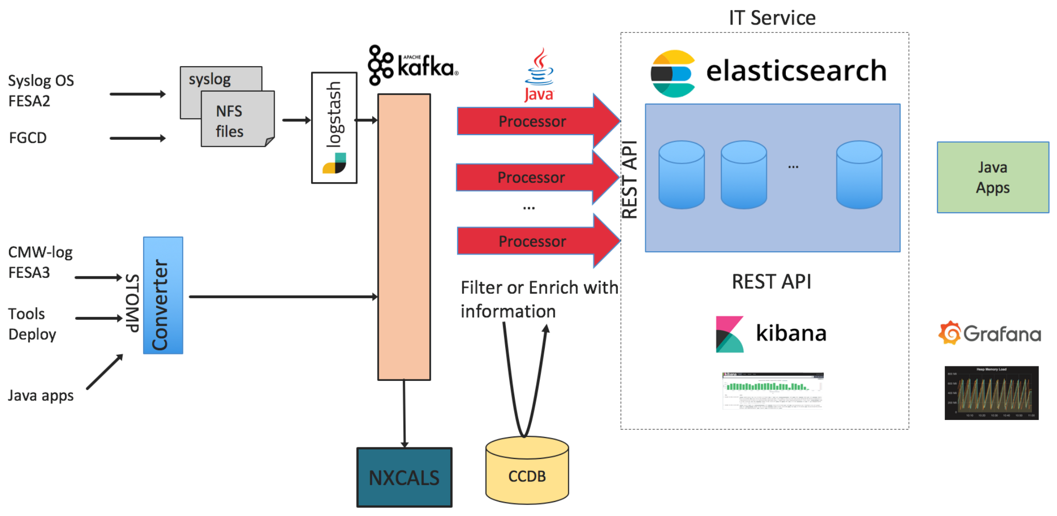
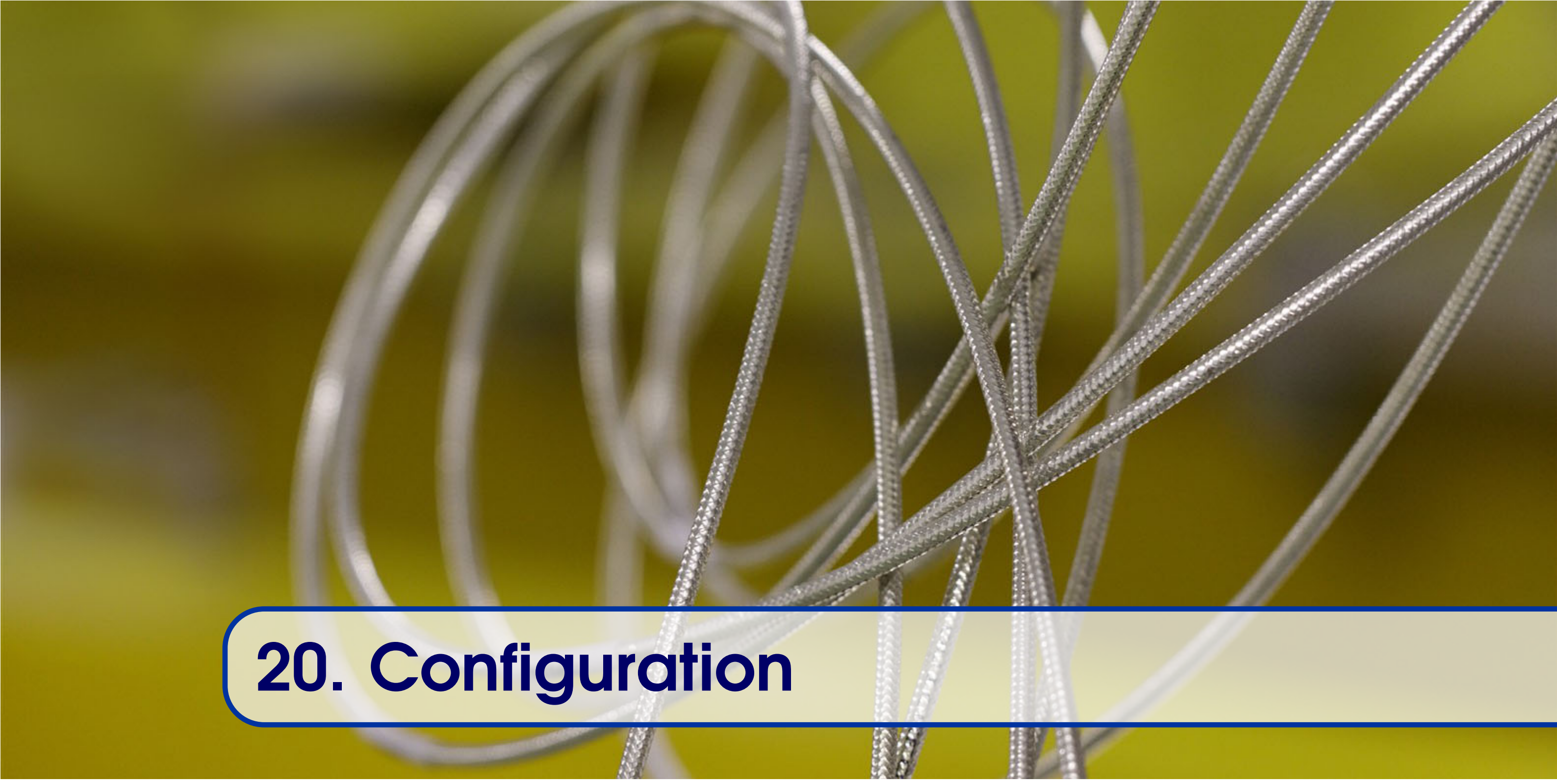


Figure 19.6: Tracing architecture

For Elasticsearch and Kibana, we rely on the services of the IT department, who have provided an up-to-date stack with high-availability, redundancy, security and expert support on a cluster on the TN.

Using this architecture with a Kafka broker, opens the door to future integration with NXCALS (see section 15.3), as a secondary data store. This would provide the means to analyse tracing data, together with accelerator data already stored in NXCALS.

This new service was deployed for the start of LS2, replacing an obsolete system and resolving several shortcomings relating to reliability, scalability, and maintainability.



20. Configuration

As described in the previous chapters, the Control System is made of many components that are generic and can be configured according to their context. In order to avoid having dispersed configuration strategies, including different technical implementations, such as text files, local databases etc. it was decided to put in place a Controls Configuration Service (CCS) [9]. The CCS aims to avoid redundancy and duplication of configuration data, as far as possible. It is composed of three main areas, the Controls Configuration Database (CCDB), the Controls Configuration Data Editor (CCDE) and various APIs including the Controls Configuration Data Access (CCDA).

20.1 Controls Configuration Database

At the centre of the CCS lies a relational database, the Controls Configuration Database (CCDB). Using a relational database brings many advantages. In addition to centralising data, which reduces redundancy and duplication of configuration data, it prevents inconsistencies and incoherencies between the configuration of components [65]. It also provides easy data management with features such as access controls, history, etc. Finally, the CCDB makes data available to any user and component of the Control System that needs it.

However, some components, such as fixed displays (see section 17.1.3), do not rely on the CCDB to store their configuration data. Sometimes this choice was deliberate, in order to produce a standalone solution. Nevertheless, these cases need to be carefully managed in order to avoid problems such as incoherent configurations. A typical case is when a device name is used as a configuration identifier in a file and it is centrally changed in the CCDB. The configuration file becomes out-of-date and the application using it will fail.

The CCDB models most of the core concepts and services found in the Control System, along with the relationships between them. For example, devices and their relationships are described in the database, allowing a better understanding of how elements in the Control System work together. In 2019, the Control System is represented in the CCDB using an estimated six-hundred and fifty domain entities. There is a high level of normalisation at the level of the database, which explains the high number of tables. The model covers the lowest level details of the Control System, such as the hardware module description (see section 12.1), to high level elements, such as the configuration of the menu items in the CCM.

The CCS has existed for over 30 years [16] and is constantly evolving due to changing requirements in the Control System and technological advances. This requires a substantial effort in order to control the technical debt and limit unnecessary complexity, whilst ensuring overall system stability [11, 46]. In 2019, the CCDB is based on an Oracle database cluster (acccon) which offers a stable and reliable service.

The two next chapters cover the recommended means of accessing data in the CCDB, which are the CCDE and the CCDA API. These interfaces hide the database implementation details and ease the evolution of the database model. Nevertheless, several users still have satellite accounts, allowing direct SQL access to views and the ability to call PL/SQL packages containing service-specific business logic.

20.2 Controls Configuration Data Editor

The CCDE is the most intuitive, non-programmatic way to access and edit configuration data, thanks to its interactive web interface. It is comprised of several modules, each focusing on an aspect of configuration. The modules are related and it is easy to navigate seamlessly between them, thus hiding complexity and implementation details from the user. For example, from the FESA instantiation editor, the user can easily access the front-end start-up sequence editor.

The main challenge when designing such a complex GUI is usability. Indeed, the many concepts modelled in the database need to be exposed, together with the relationships between them in a integrated way. At the same time, the different workflows of the diverse user community, need to be taken into account in order to facilitate the work of the users. This topic of usability and how it is applied in the design of the CCDE is further explored in [12].

The CCDE is based on the ACW stack (see section 17.1.4). As such, the back-end is written in Java and the main framework used on the client-side is Angular JS. To guarantee high-availability, the back-end is deployed on a cluster of two machines and is load-balanced using HAProxy. This architecture is depicted in figure 20.1.

Compared to the previous APEX-based solution [64], this new architecture has many advantages. It is 3-tier and therefore offers good decoupling between the client-side and the database, and since there is a Java server available, interaction with other systems is possible e.g. calling a Python script to generate files. Furthermore, it relies on open-source components and reduces vendor lock-in with Oracle.

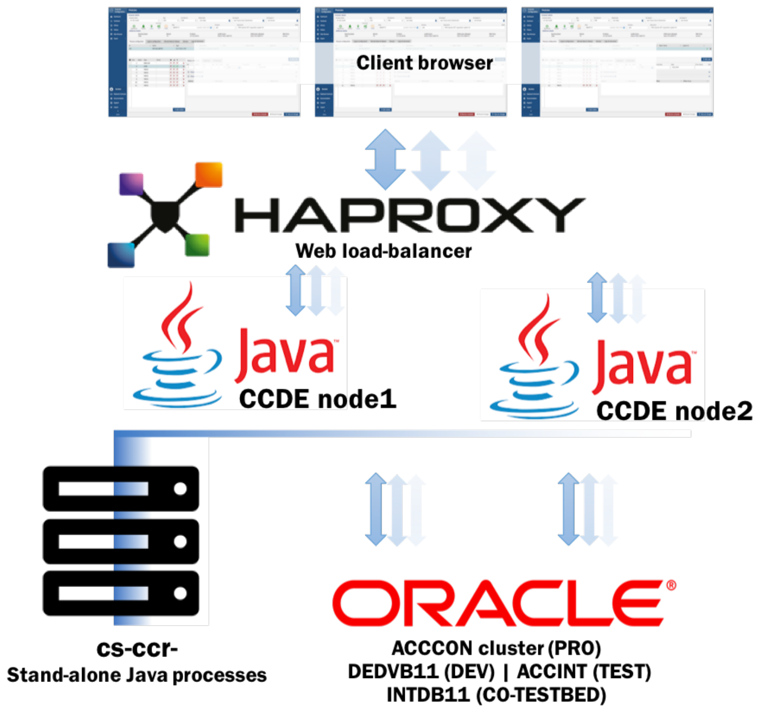


Figure 20.1: CCDE architecture

20.3 Controls Configuration Data Access API

Providing user-friendly GUIs is an absolute necessity, but in an environment such as CERN it is not sufficient, since very often users require programmatic access to data. The Controls Configuration Data Access (CCDA) API fulfils this need.

Developed as part of the LS2 programme to replace an obsolete service, the so-called `configdb-dirservice`, the CCDA provides access to the configuration data through a REpresentational State Transfer (REST) API. This was also a good opportunity to review and remove obsolete configuration data used by deprecated frameworks such as GM.

REST has been chosen, as it is a language agnostic way to expose an API. In the past, the recommended approach was to use RMI, which is Java specific, but with the increase in popularity of other languages, such as Python, an API with the ability to work with several languages was required. The CCDA relies on an architecture similar to the one used for the CCDE, (see figure 20.2), which is based on a Java server running on two separate computers, load-balanced using HAProxy.

To improve the user-friendliness of the API, a Java wrapper is also available to hide the low-level implementation details of the REST endpoints. The Java wrapper offers easy access to domain objects representing the core concepts of the controls configuration. In 2019, there are plans to provide wrappers in other languages, such as C++ or Python. In addition, the REST endpoints can be used directly and wrappers developed by the user community might emerge.

Like its predecessor, the first version of the CCDA, released in 2019, is read-only. However, there are plans to provide read-write access in order to allow the development of

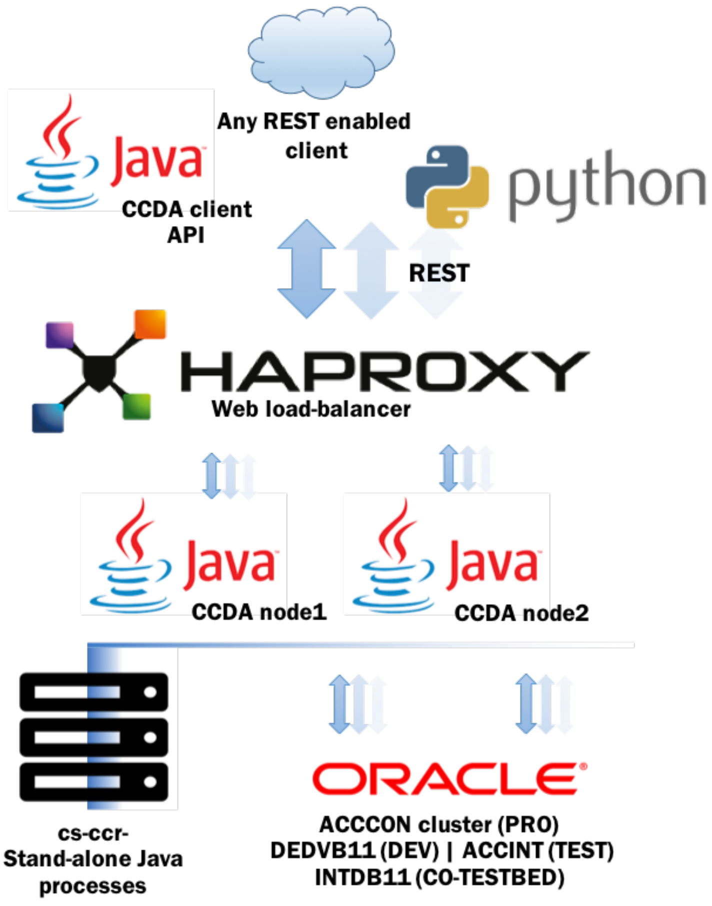
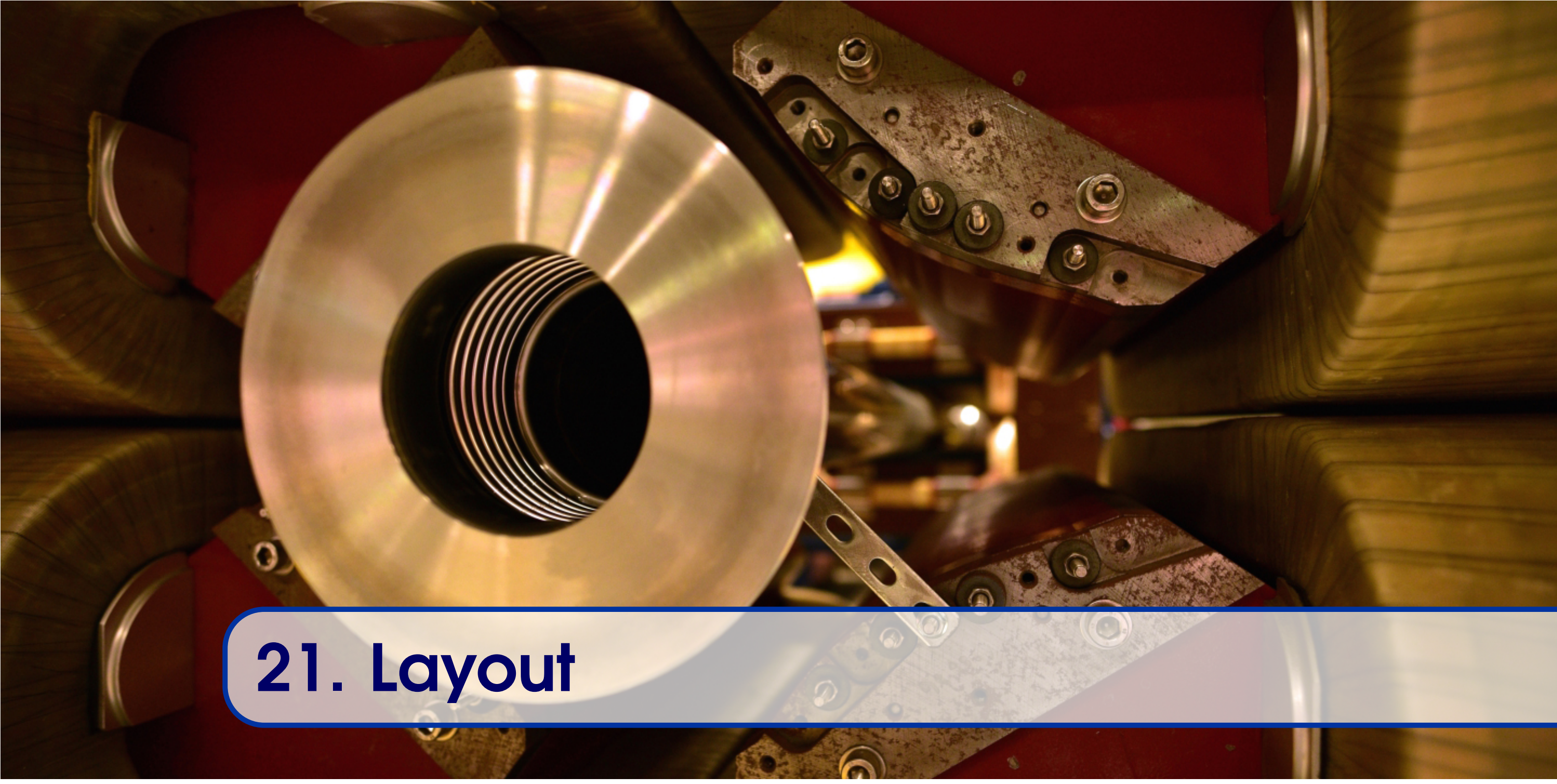


Figure 20.2: CCDA architecture

more powerful tooling and better integration with other systems. Read-write access will enable complex workflows to be modelled, which would require interaction with several components.

VII

Data Management



21. Layout

The Layout Database and its initial tools were originally developed in 2003 as a joint collaboration between the Controls group and the Installation and Commissioning group, as a tool for planning the installation of beam line components in the LHC [56].

One of the key concepts of the Layout Database is that hardware components are described as Functional Positions (FPs). The combined functional position data from more than 20 equipment groups forms a centralised, integrated, cross-domain model of the physical installations in the accelerator complex. By centralising and sharing data that would otherwise be stored in dispersed, private, domain-specific databases, the equipment groups profit from cross-domain maintenance i.e. automatic updates induced by other group's actions, as well as a coherent, consolidated dataset describing the complete architecture of all accelerators, with no redundancy.

Over the course of its lifetime, the purpose, geographical scope and domain scope of the Layout Service has expanded enormously; from just managing LHC beam-line elements, to potentially managing any functional position that has an impact on accelerator operation at CERN [55]. The Layout database has been used for a variety of purposes over time including:

- The automatic generation of MAD X sequence files for optic simulations;
- To document the complex LHC electrical circuits;
- To provide information to the transport team so that they can plan and define installation/maintenance path/trajectory of each equipment component;
- To provide links to installation planning and mounting/dismounting scenarios of a component;
- To store data for hardware testing and commissioning, in particular instrumentation for cryogenics [27, 61], vacuum and protection;
- To store data on controls objects such as electronics for the controls of the cryogenics.

A Functional Position (FP) specifies the type and name(s) of component, along with the size and the position of the space it occupies in the accelerator. Figure 21.1 shows how magnets are represented as Functional Positions in the Layout Database.

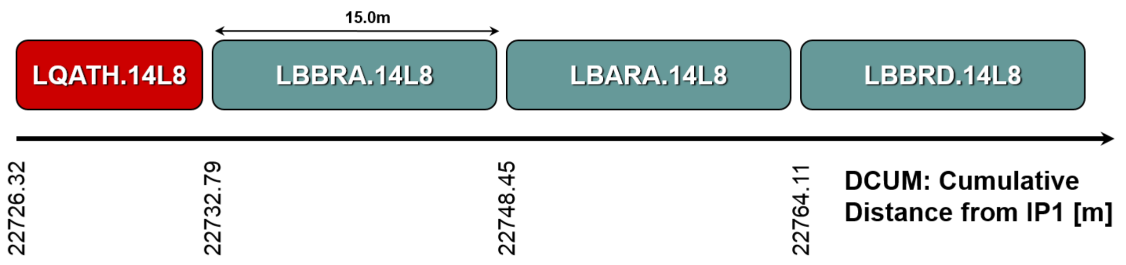


Figure 21.1: Magnet components described as Functional Positions

Functional Positions are defined as part of an Assembly Breakdown Structure (ABS). For example: An instrument FP is defined as the child of a magnet FP, as shown in figure 21.2. The position of the instrument is defined as an offset with respect to the start position of the magnet. The exact position of the instrument in the accelerator is then calculated. If the position of the magnet changes, the position of the instrument is updated automatically.

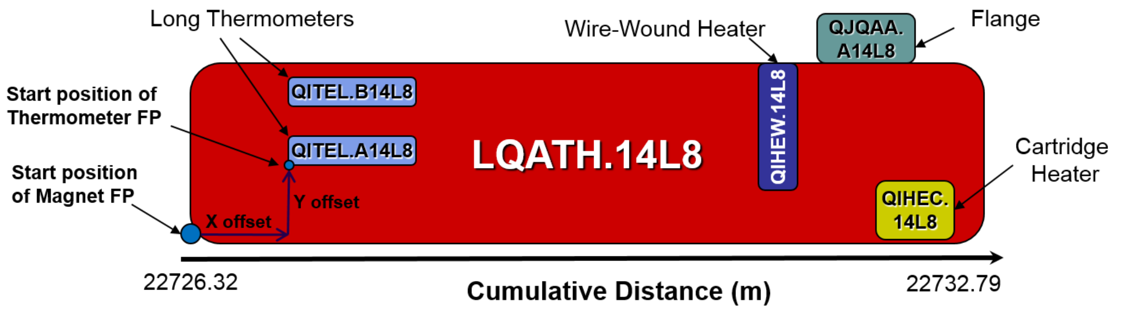


Figure 21.2: Instrumentation components described as Functional Positions within an assembly

As well as being part of one or more Assembly Breakdown Structures, individual Functional Positions can also be logically linked together in other ways, for example, to represent cabling or other types of connections.

The Layout database is also a hub linked to approximately 40 other data repositories, such as Electronic Document Management System (EDMS) for documents, InforEAM for asset management, as well as many domain-specific databases including the Cablotheque for cabling data, Sensorbase for instrument calibrations, Norma for magnet parameters and AlimDB for power converters, among others. Using these database links, the Layout web interface displays layout data aggregated with data from many other different sources to give a full picture of the installation.

The original architecture of the Layout Service used an Oracle database with a .NET web interface for browsing, as well as a set of Oracle Forms and Apex applications for editing specific subsets of data. The database schema applied an LHC-centric, generic data model, where attributes are separated from objects and stored in a separate table. This allows a large degree of flexibility when adding new properties, but has the clear disadvantage that it is not possible to implement data integrity checks on the values. Therefore, data entry and editing had to be performed by a small team of experts within the Layout Service. Long-term, this approach was not scalable and it became essential to devolve responsibility for data management to the end-users. In order to do this, the database model was consolidated and redesigned so that the data can be safeguarded by applying a detailed authorisation scheme and implementing checks which enforce domain-specific business rules, whilst at the same time allowing flexibility, modularity and extensibility. Intensive development of the new Layout Database and its new modern Graphical User Interfaces began in 2014. It comprises of a new Oracle Database and a unified read-write web interface built using the ACW framework on the common BE-CO web technology stack.

The original model was limited to archived snapshots of data at specific moments in time, plus a STUDY version which showed the current state of the installation. The new model is time-oriented; functionality required to support the management of past, current and future layouts. This allows beam physicists and operators to simulate and refine possible future machine layouts in the conext of projects such as LHC Injector Upgrade (LIU), High-Luminosity Large Hadron Collider (HL-LHC), and Future Circular Collider (FCC). In addition equipment groups are able to communicate their changes in advance, thus allowing a smoother distribution of Layout activities over time and less labour intensive work at the end of stop periods. However, this new functionality is complex and pushes the current capabilities of Oracle's referential integrity management system to its limits.

22. Organising Accelerator Operation

Several aspects of the accelerator's operation have to be organised, such as the schedule, controls interventions, and follow-up of issues encountered and how they affect the availability of the machines. BE-CO provides two tools, Accelerator Fault Tracking (AFT) and Accelerator Schedule Management (ASM) to assist with these tasks.

22.1 AFT

One of the prevailing goals of the Accelerator and Technology Sector is to optimise the efficiency of the accelerators in terms of protons produced and luminosity delivered to the experiments. Therefore, it is important to be able to identify the root causes of downtime over time, in order to in-turn, prioritise the corresponding consolidation work [4, 52].

The Accelerator Fault Tracking (AFT) initiative aims to provide the infrastructure necessary to consistently and coherently capture, persist, and make available accelerator fault data for further analysis. AFT serves as the main input to the work of CERN's Availability Working Group (AWG), who are tasked with providing in-depth analysis and reports of the availability of CERN's accelerators, and highlighting re-occurring problem areas to be investigated further.

The central concept of AFT is Faults, which are unplanned periods of unavailability to provide beams for physics. This may be caused by failures of physical equipment, software, human error, or naturally occurring effects such as beam losses (e.g. LHC Unidentified Falling Objects (UFOs)).

AFT provides various means to structure the data. Faults are categorised in systems according to their cause (e.g. Cryogenics, Vacuum, Accelerator Controls etc.). Systems are not tied to CERN administrative units, which are subject to change over time. Two or more faults can be linked using fault relations, which also describe the nature of the link (e.g. parent-child, "same as", "similar to", "related to").

Based on the fault data, AFT computes Availability and Downtime. These complementary notions measure the time that a system was available (or unavailable). This can be seen from different perspectives according to the wishes of the AFT user. For example, a system expert may be interested in "raw" system unavailability, whereas as people planning consolidation may be interested in "root cause" unavailability, taking into account fault relations such as "parent-child" to uncover dependencies and target the consolidation of root causes rather than symptoms.

Faults are registered by the operators of the different accelerators via the ELogbook application. Behind the scenes, the fault is directly registered in AFT. Once a week, the Availability Working Group (AWG)'s members and Machine Supervisors meet, together with an AFT expert, to review fault assignments and data completeness and correctness. Following this review, all faults are marked as being "Reviewed by AWG". When a fault is assigned to a system, the corresponding expert is notified and is able to update and complete certain fault attributes, using the AFT Web application. If they agree with the fault assignment, they can mark the fault as being "Reviewed by Expert", otherwise they can request that certain controlled attributes (system assignment, start and end times) are modified by the AWG. A summary of the accelerator performance, based on AFT, is presented at daily and weekly operational review meetings, as well as more in-depth reports following each accelerator Technical Stop, and at the end of the annual physics run.

AFT is comprised of a database, Web application and underlying APIs and processes to store, manage and analyse all data related to faults. AFT is based on the common ACW technology stack used in BE-CO for Web application development (see section 17.1.4). Figure 22.1 shows an AFT dashboard for the LHC, including the overall availability for the selected time period, and the so-called "LHC Cardiogram".

The AFT project was launched at the end of February 2014, and delivered a production ready system at the start of 2015, to capture LHC fault data from an operational perspective. Following its success during 2015/16, the system was extended to cover the complete Injector Complex in 2017. In addition, further iterations of the AFT system have provided an increasing amount of functionality to support fault management, follow-up by equipment groups (in addition to the operations teams) and analysis by all users. In 2019, development continued on many aspects, including integration with other systems, such as Accelerator Schedule Management (ASM) (see section 22.2) and Layout (see chapter 21), providing additional attributes and improved reporting and analysis functionality.

In the future, further integration with other systems, such as LASER (see section 15.7), Logging (see section 15.3), and the CCS (see chapter 20), is foreseen. Other possible improvements include active data analysis, predictive failure analysis using advanced algorithms and/or Machine Learning, looking for failure patterns, and combining fault data with logged data, where applicable, to detect failures.

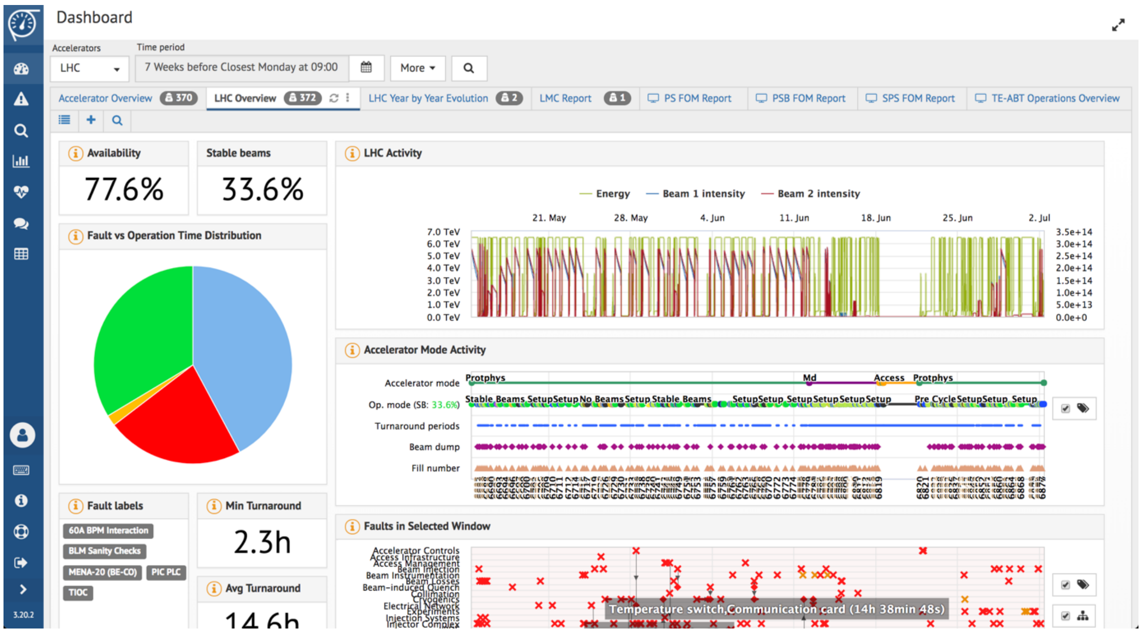


Figure 22.1: AFT dashboard for the LHC

22.2 ASM

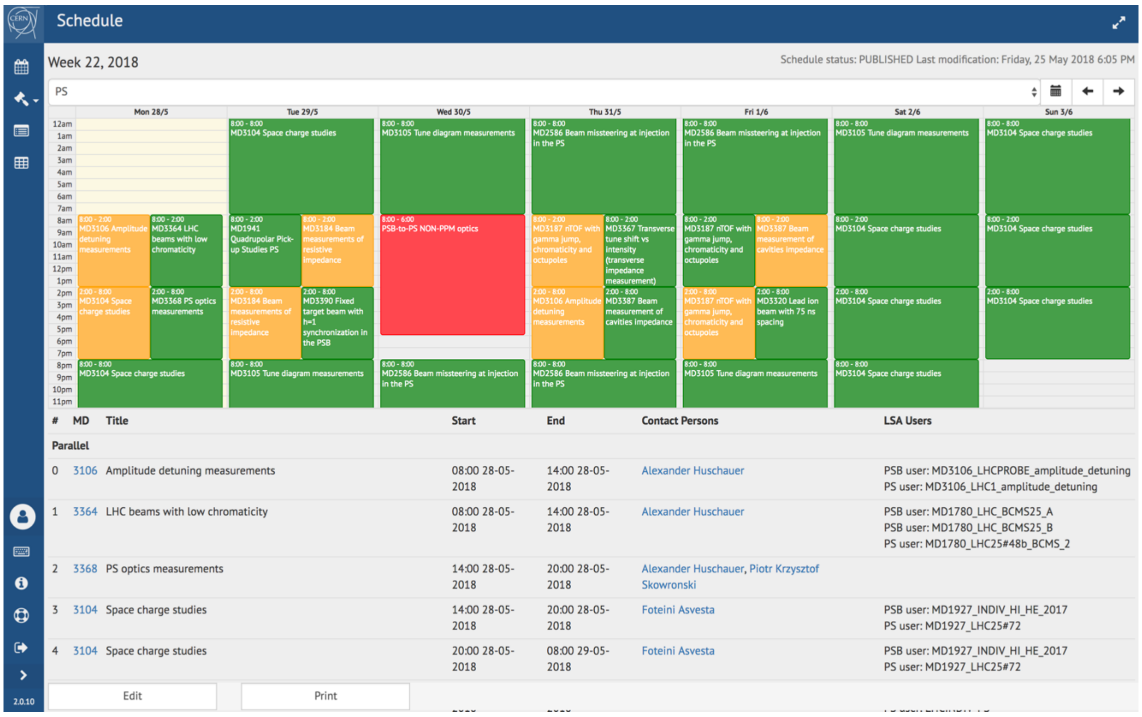
Operating CERN's accelerator complex requires careful forward planning and synchronised scheduling of common events across the machines, such as Technical Stops, MD's and special physics runs. The schedules are of interest to many people, helping them to plan and organise their work and perform different data analyses. Therefore, this data should be easily accessible, both interactively and programmatically. The ASM system aims to address these topics, by allowing users to centralise the definition of schedules and events assigned to those schedules.

ASM provides the infrastructure necessary to define, manage and publish schedule data in a generic way. It allows future work to be planned according to scheduled events, in a way that entries remain valid, even if the actual dates of the scheduled events are changed e.g. a plan to upgrade a FESA device class during a technical stop remains valid, even if the date of the technical stop changes. The credibility of the application is fully dependent on the correctness of the schedule data, which requires regular data entry and updates by the responsible person.

Schedules and events are configured in a data driven manner, making it relatively simple to add support to manage different types of schedules. As depicted in figure 22.2, schedules and events are managed and consulted using the ASM Web application which is based on the common ACW technology stack (see section 17.1.4).

A REST API also provides programmatic access to the schedule data, enabling ASM to be easily used as input to other systems, such as AFT (see section 22.1), or potentially Logging (see section 15.3). Typical use cases could be to display LHC faults between TS1 and TS2 in 2016, or to extract logged BPM data during the LHC MD block 1 in 2018.

The ASM project was launched in February 2017, initially focusing on yearly accelerator



Schedule

Week 22, 2018

Schedule status: PUBLISHED Last modification: Friday, 25 May 2018 6:05 PM

PS

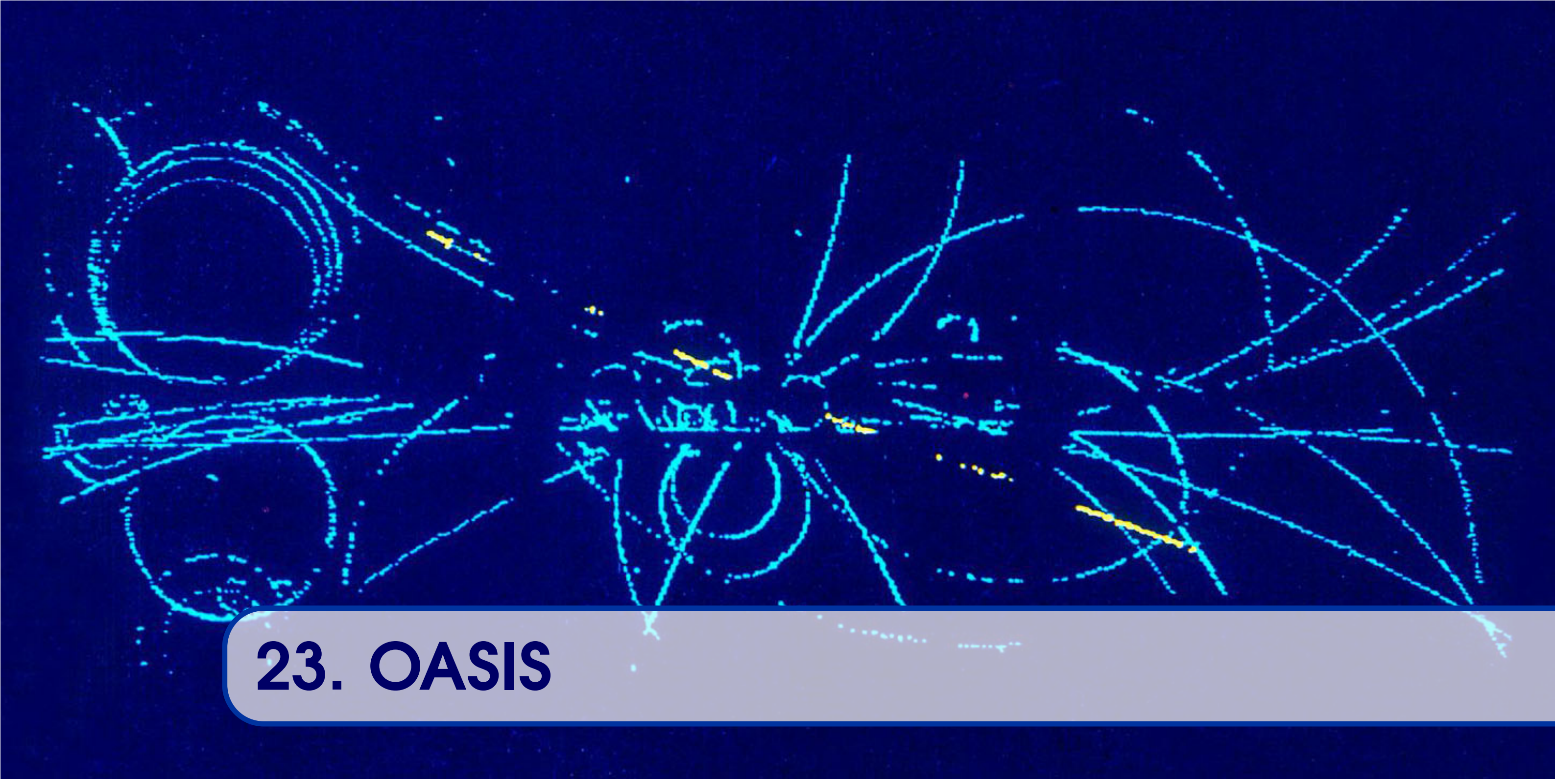
Mon 28/5	Tue 29/5	Wed 30/5	Thu 31/5	Fri 1/6	Sat 2/6	Sun 3/6
	8:00 - 8:00 MD3104 Space charge studies	8:00 - 8:00 MD3105 Tune diagram measurements	8:00 - 8:00 MD2586 Beam missteering at injection in the PS	8:00 - 8:00 MD2586 Beam missteering at injection in the PS	8:00 - 8:00 MD3105 Tune diagram measurements	8:00 - 8:00 MD3104 Space charge studies
8:00 - 2:00 MD3106 Amplitude detuning measurements; 8:00 - 2:00 MD3364 LHC beams with low chromaticity	8:00 - 2:00 MD1941 Quadrupolar Pick-up Studies PS; 8:00 - 2:00 MD3184 Beam measurements of resistive impedance	8:00 - 6:00 PSB-to-PS NON-PPM optics	8:00 - 2:00 MD3187 nTOF with gamma jump, chromaticity and octupoles; 8:00 - 2:00 MD3367 Transverse tune shift vs intensity (transverse impedance measurement)	8:00 - 2:00 MD3187 nTOF with gamma jump, chromaticity and octupoles; 8:00 - 2:00 MD3387 Beam measurement of cavities impedance	8:00 - 2:00 MD3104 Space charge studies	8:00 - 2:00 MD3104 Space charge studies
2:00 - 8:00 MD3104 Space charge studies; 2:00 - 8:00 MD3368 PS optics measurements	2:00 - 8:00 MD3184 Beam measurements of resistive impedance; 2:00 - 8:00 MD3390 Fixed target beam with h=1 synchronization in the PSB		2:00 - 8:00 MD3106 Amplitude detuning measurements; 2:00 - 8:00 MD3387 Beam measurement of cavities impedance	2:00 - 8:00 MD3187 nTOF with gamma jump, chromaticity and octupoles; 2:00 - 8:00 MD3320 Lead ion beam with 75 ns spacing	2:00 - 8:00 MD3104 Space charge studies	2:00 - 8:00 MD3104 Space charge studies
8:00 - 8:00 MD3104 Space charge studies	8:00 - 8:00 MD3105 Tune diagram measurements	8:00 - 8:00 MD2586 Beam missteering at injection in the PS	8:00 - 8:00 MD2586 Beam missteering at injection in the PS	8:00 - 8:00 MD3105 Tune diagram measurements	8:00 - 8:00 MD3104 Space charge studies	

#	MD	Title	Start	End	Contact Persons	LSA Users
Parallel						
0	3106	Amplitude detuning measurements	08:00 28-05-2018	14:00 28-05-2018	Alexander Huschauer	PSB user: MD3106_LHCPROBE_amplitude_detuning PS user: MD3106_LHC1_amplitude_detuning
1	3364	LHC beams with low chromaticity	08:00 28-05-2018	14:00 28-05-2018	Alexander Huschauer	PSB user: MD1780_LHC_BCMS25_A PSB user: MD1780_LHC_BCMS25_B PS user: MD1780_LHC25#48b_BCMS_2
2	3368	PS optics measurements	14:00 28-05-2018	20:00 28-05-2018	Alexander Huschauer, Piotr Krzysztof Skowronski	
3	3104	Space charge studies	14:00 28-05-2018	20:00 28-05-2018	Foteini Asvesta	PSB user: MD1927_INDIV_HI_HE_2017 PS user: MD1927_LHC25#72
4	3104	Space charge studies	20:00 28-05-2018	08:00 29-05-2018	Foteini Asvesta	PSB user: MD1927_INDIV_HI_HE_2017 PS user: MD1927_LHC25#72

Figure 22.2: PS Machine Development schedule for week 22 of 2018

schedules for the Injector complex and LHC. Shortly afterwards, the system was extended to provide integrated MD management, as well as the registration, approval and follow-up of Controls changes.

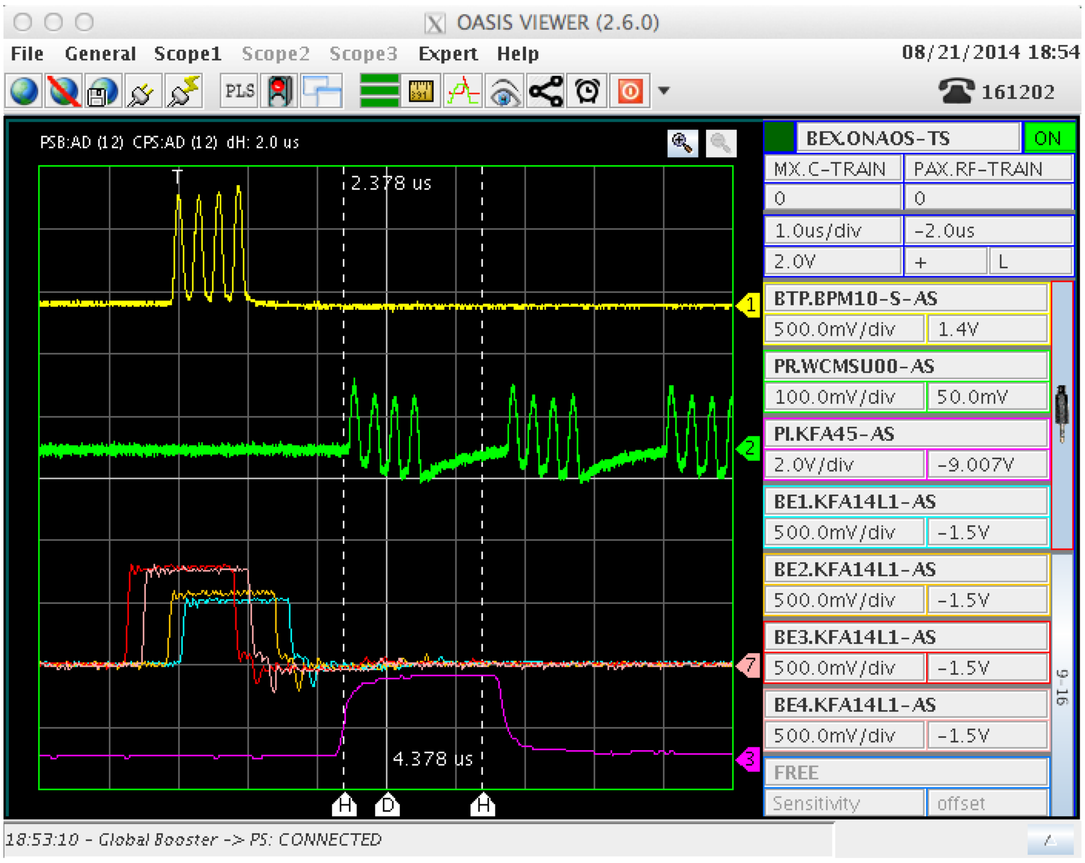
VIII Control System Applications



23. OASIS

Operators and system experts need to acquire low-level analogue signals coming from the different pieces of equipment. The sources of the signals are scattered around the accelerators, but the users need to easily correlate them, regardless of their relative distances. For example, in the PS complex, it is not unusual to have hundreds of metres between sources that need to be observed together. To cover this need, the OASIS project was launched in the early 2000s [21, 22], with the aim to provide a new analogue signal acquisition infrastructure for the LHC, and to replace the existing system in the other accelerators. The core concept of OASIS is the virtual oscilloscope, through which we can display signals acquired in different locations on the same oscilloscope screen, see figure 23.1, whilst guaranteeing a time coherency. The time coherency is ensured thanks to a single central trigger generation system. The pulses are then distributed via direct cables between the trigger generator and the digitisers, installed around the accelerators. However, since there are thousands of signals to be observed, and it would be economically prohibitive to install a digitiser for every signal, therefore OASIS uses signal multiplexing wherever possible. A switch matrix accepting a large number of input signals is installed before a group of digitisers, allowing the users to select which signal to observe at a given moment. The number of signals that can be observed is still limited to the quantity of digitisers, but overall the amount of available signals is higher. Finally, to complete the virtual oscilloscope illusion, OASIS has to manage the various digitiser settings in a coherent way to guarantee that what is displayed makes sense. For example, if one changes the trigger delay in the virtual oscilloscope, the appropriate delay values have to be sent to the different digitisers.

For its implementation OASIS relies as much as possible on the BE-CO building blocks. In terms of hardware, we strive to stay in-line with the supported BE-CO front-end platforms, as described in chapter 5. The first generation of OASIS hardware reused the existing VXI crates and introduced the CompactPCI format. With the introduction of industrial PCs



BEX.ONAOS-TS ON
MX.C-TRAIN PAX.RF-TRAIN
0 0
1.0us/div -2.0us
2.0V + L

BTP.BPM10-S-AS 500.0mV/div 1.4V
PR.WCMSU00-AS 100.0mV/div 50.0mV
PI.KFA45-AS 2.0V/div -9.007V
BE1.KFA14L1-AS 500.0mV/div -1.5V
BE2.KFA14L1-AS 500.0mV/div -1.5V
BE3.KFA14L1-AS 500.0mV/div -1.5V
BE4.KFA14L1-AS 500.0mV/div -1.5V

18:53:10 - Global Booster -> PS: CONNECTED

Figure 23.1: Virtual oscilloscope showing the transfer for PS Booster to PS

in the Control System and their PCI and PCIe buses, new generations of digitisers were integrated into the system and provided an opportunity to eradicate VXI. Furthermore, a VME solution was developed for low-bandwidth, low cost installations. In the future, the CompactPCI platform will be phased out and new platforms such as mTCA will be supported.

The FEC software to perform the RT control of the hardware modules e.g. digitisers, signal matrices and trigger generators, also relies on standard BE-CO solutions, namely FESA (see section 11.2). The device-property interfaces for the different hardware types have been specified and documented to allow equipment groups to integrate new types of digitisers, which are not yet supported, into OASIS's front-end layer [20]. Typical front-end interfaces expose settings to be controlled by the higher layers. However, in the case of OASIS, in order to easily support various types of hardware, the interface also exposes the capabilities of the underlying piece of hardware. For example, the maximum number of samples depends on the on-board memory and other parameters, such as the timebase etc. Therefore, the corresponding property will publish the current number of samples, as well as the maximum number of samples available with the current set-up.

In the middle tier, the OASIS server, implemented in Java, is in charge of the management of resources and settings. The resources are the digitisers and signal matrices that have to be assigned to the different clients, depending on several criteria, such as their location and user name, in order to maximise the number of signals that can be observed at a given time.

To perform the signal routing, the server relies on device relationships stored in the CCDB (see section 20.1). Figure 23.2 depicts the front-end interfaces with their relationships. The virtual oscilloscope illusion is obtained thanks to the OASIS setting management. In a given virtual oscilloscope, the different signals might be acquired by hardware modules with different characteristics. As well as ensuring that compatible settings are used for the digitalisation, the middle tier also has to compute the intersection of the capabilities of the different digitisers. For example, on a virtual oscilloscope with two signals, one digitiser might have twice the available pre-trigger memory than the other and the OASIS server has to limit the maximum delay that will be compatible with the two modules. The communication with the GUI is based on RMI and JMS, as for any systems developed in the early 2000s.

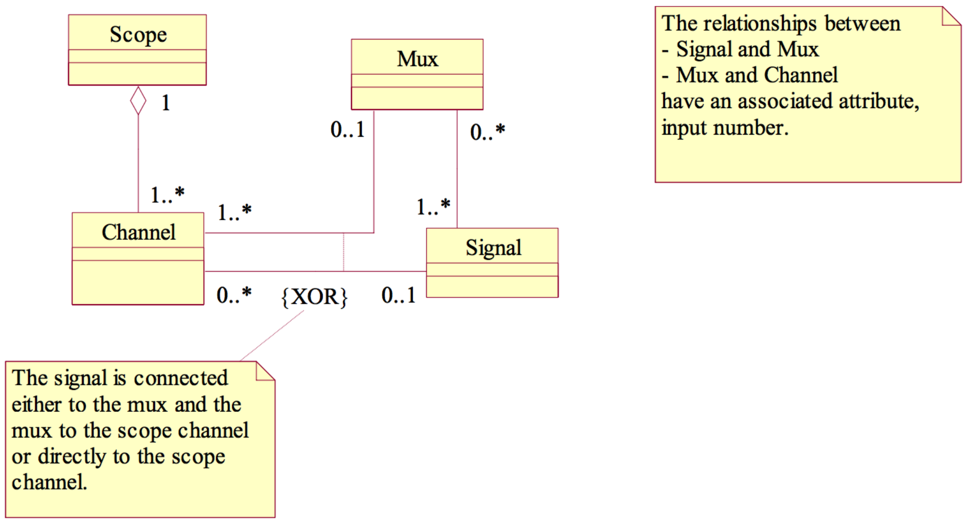
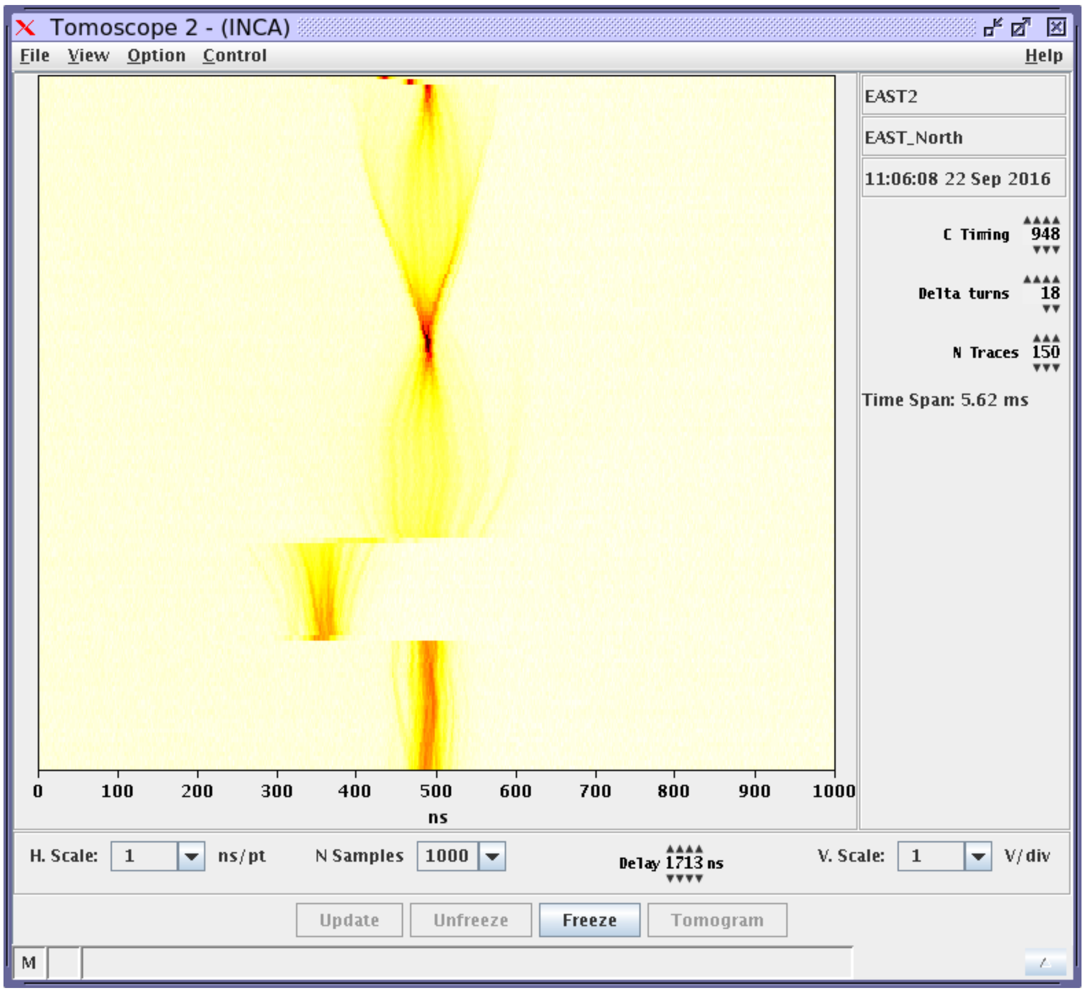

Scope
1
Mux
0..1
0..*
1..*
1..*
1..*
Channel
Signal
0..*
{XOR}
0..1
The relationships between
- Signal and Mux
- Mux and Channel
have an associated attribute,
input number.
The signal is connected either to the mux and the mux to the scope channel or directly to the scope channel.

Figure 23.2: OASIS front-end model

At the top of the stack, there is a generic viewer offering three 16-channel virtual oscilloscopes, allowing users to connect any signals available in OASIS. There is also a Java client library available for users to develop their own specific graphical clients, such as the tomoscope application shown in figure 23.3. The generic viewer was developed in Java using Swing and particular attention was paid to ensure the high refresh rate required for display modes such as scrolling.

The biggest problem for the future extension of OASIS is the way that triggers are distributed. Indeed, whenever a new installation is made, cables, potentially hundreds of metres long, have to be pulled from the trigger generation crate to the new digitisers. For large machines such as LHC and SPS, this is often impractical. Therefore, signals acquired in different points of the accelerators cannot be displayed on the same oscilloscopes due to the lack of coherent triggering. The future plan is to deploy a White Rabbit network to distribute the triggers and to rely on the WRTD specification, as described in section 5.6. The challenge with this new distribution to ensure that the delay, which needs to be added to a trigger, is compatible with the hardware, taking into account the network latency and the memory available in the digitisers to store pre-trigger samples at the required sampling frequency.



EAST2

EAST_North

11:06:08 22 Sep 2016

C Timing 948

Delta turns 18

N Traces 150

Time Span: 5.62 ms

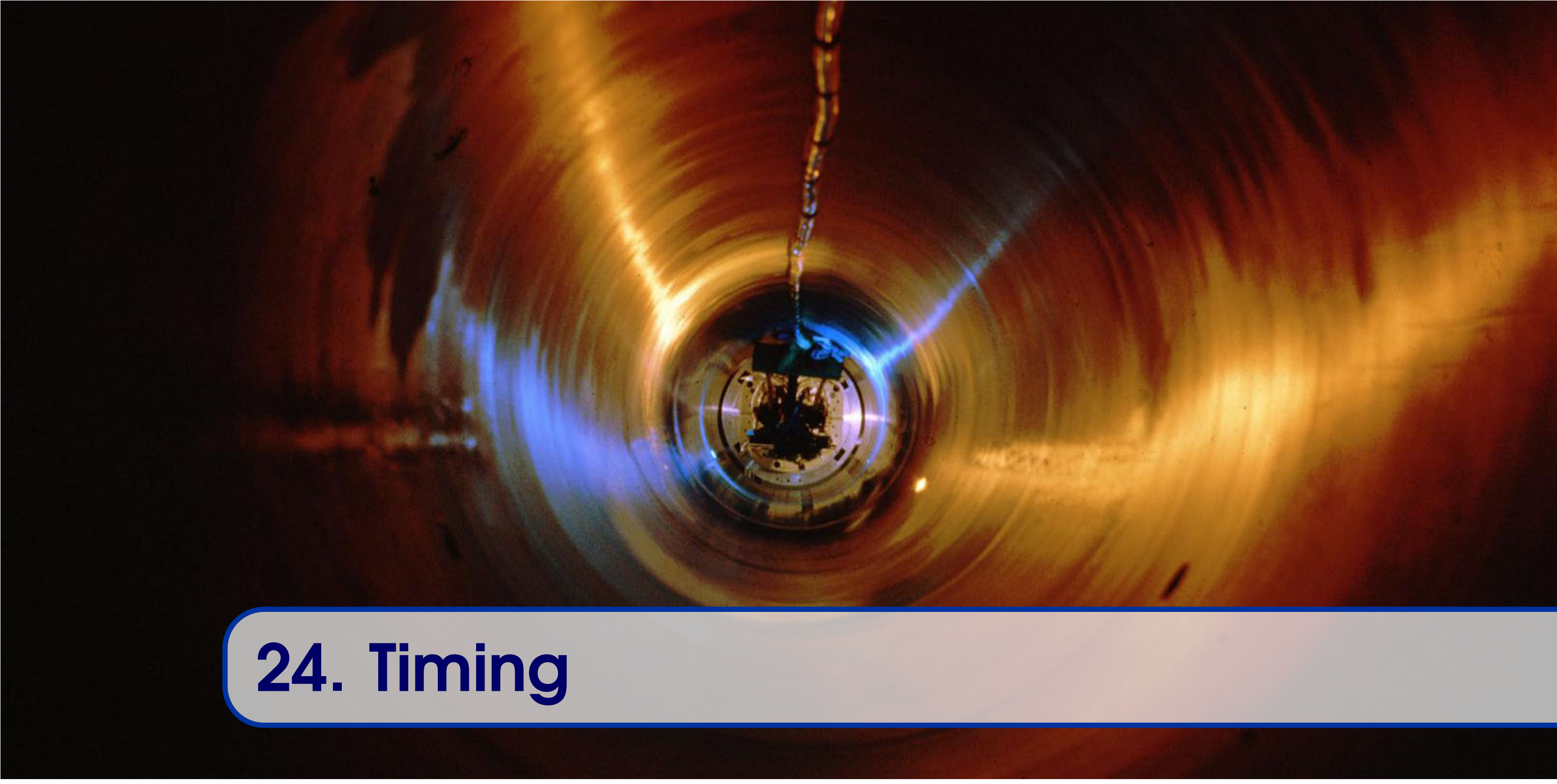
H. Scale: 1 ns/pt

N Samples 1000

Delay 1713 ns

V. Scale: 1 V/div

Figure 23.3: Tomoscope application relying on OASIS infrastructure to acquire pickup signals



24. Timing

As introduced in chapter 4, the Timing system is at the core of the accelerator controls. Figure 24.1 depicts its main components. The Central Timing (CT) takes Beam Coordination Diagrams (BCDs) from the operators and, taking into account the external condition inputs, schedules the cycles for the different accelerators accordingly. Events and contextual information, known as the Telegram, are sent on the GMT network and received by the Central Timing Receivers (CTRs) that in turn decode the contextual information, generate interrupts and pulses, as well as derived events resynchronised with accelerator clocks, such as the beam revolution clock.

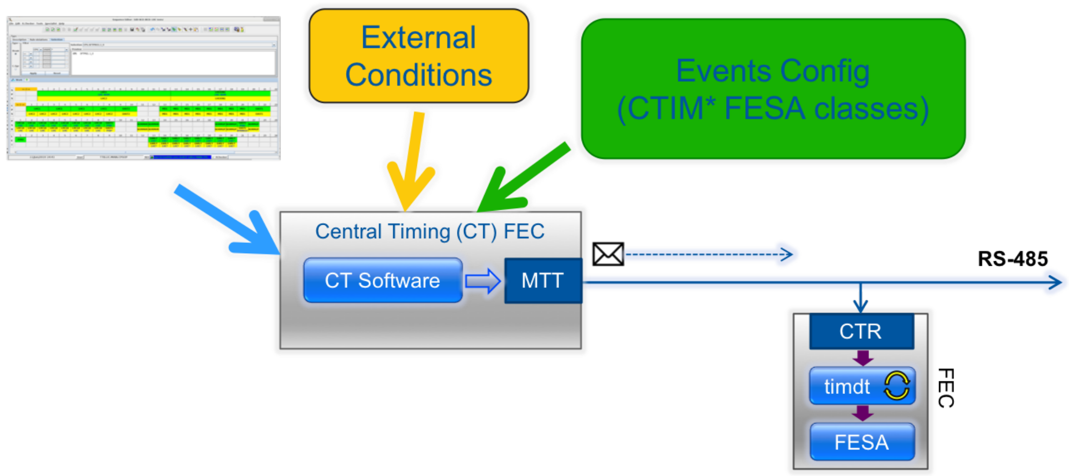


Figure 24.1: Overall architecture of the Timing system

24.1 Central Timing

The Central Timing Front End software is based on five FESA classes (see section 11.2). These five classes communicate with each other thanks to FESA association and are deployed on the central timing FEC as a single FESA deploy-unit.

One FESA class is in charge of collecting external condition information. External conditions are used to communicate problems with critical devices such as power converters etc. or beam requests/inhibits to the central timing. There are two types of input: hardware and software. For the hardware inputs, there is a network of PLCs that collect the hardware conditions and the FESA class reads them using SILECS (see section 12.2). The software conditions are obtained directly by subscribing to the device using CMW.

Three other FESA classes provide the central event control (delay, enable/disable, etc.) and the last FESA class implements the central timing scheduling logic, thanks to the information provided by the other classes and the BCD programmed by the operators.

Once the scheduling is complete, messages are distributed using one Multi-Tasking Timing (MTT) generator [1] per timing domain (see section 4.3) after resynchronisation with the Global Positioning System (GPS) signals. As shown in figure 24.2, a GPS receiver is used to generate three stable synchronisation signals: a Pulse-Per-Second (PPS), a 40MHz clock, and a 1.2-second-period signal. The latter, known as the basic period, is the heartbeat of the accelerator complex.

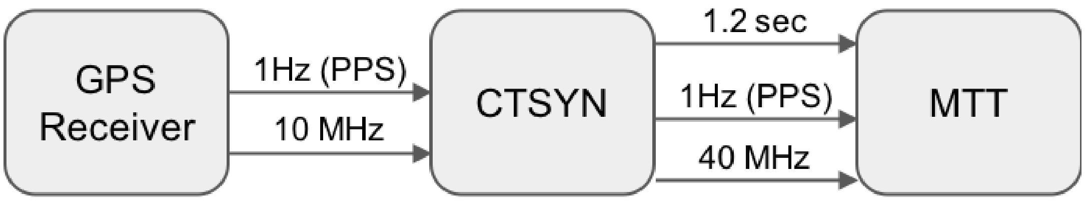


Figure 24.2: Central Timing synchronisation signals

The main GUI, the Sequence Editor, is used by the accelerator operators to program the beams to be played, along with the spare beams. This application was developed in Java Swing and communicates with a central Java server using RMI. Figure 24.3 shows the BCD editor with the supercycles from LEIR to SPS with the normal beams in green and the spare beams in yellow.

24.2 Distributed Timing

The timing distribution, called the GMT network, is based on the RS485 standard and encodes the messages using Manchester code. This technology was deployed more than 30 years ago and has very limited bandwidth, allowing the central timing to send only eight 32-bit frames per millisecond. Due to this limitation, only one timing domain can be handled by a single MTT generator and therefore the central timing front-end contains several modules.

When the GMT signal needs to be transported over a long distance, a conversion to optical signals is performed using modules designed specifically for the GMT distribution. This

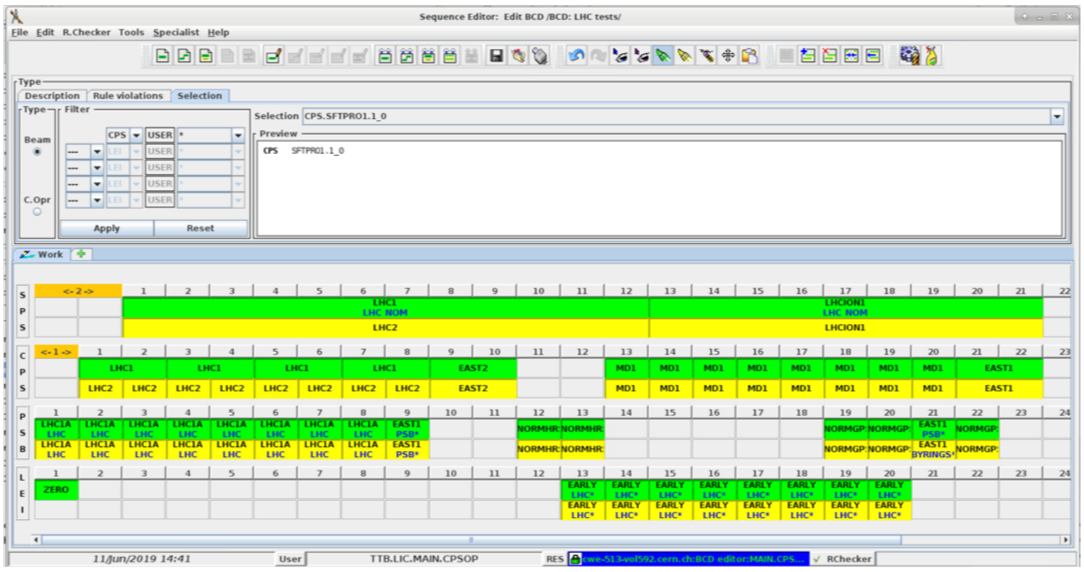


Figure 24.3: BCD editor

is typically used between CERN sites (Meyrin, Prevessin) and SPS and LHC points. Inside a technical building, the optical signal is converted back into an electrical signal and distributed over a copper network with GMT repeaters and fan-outs when required. Like the MTT generator, the GMT repeaters are VME modules and VME RTMs (see section 5.3.1).

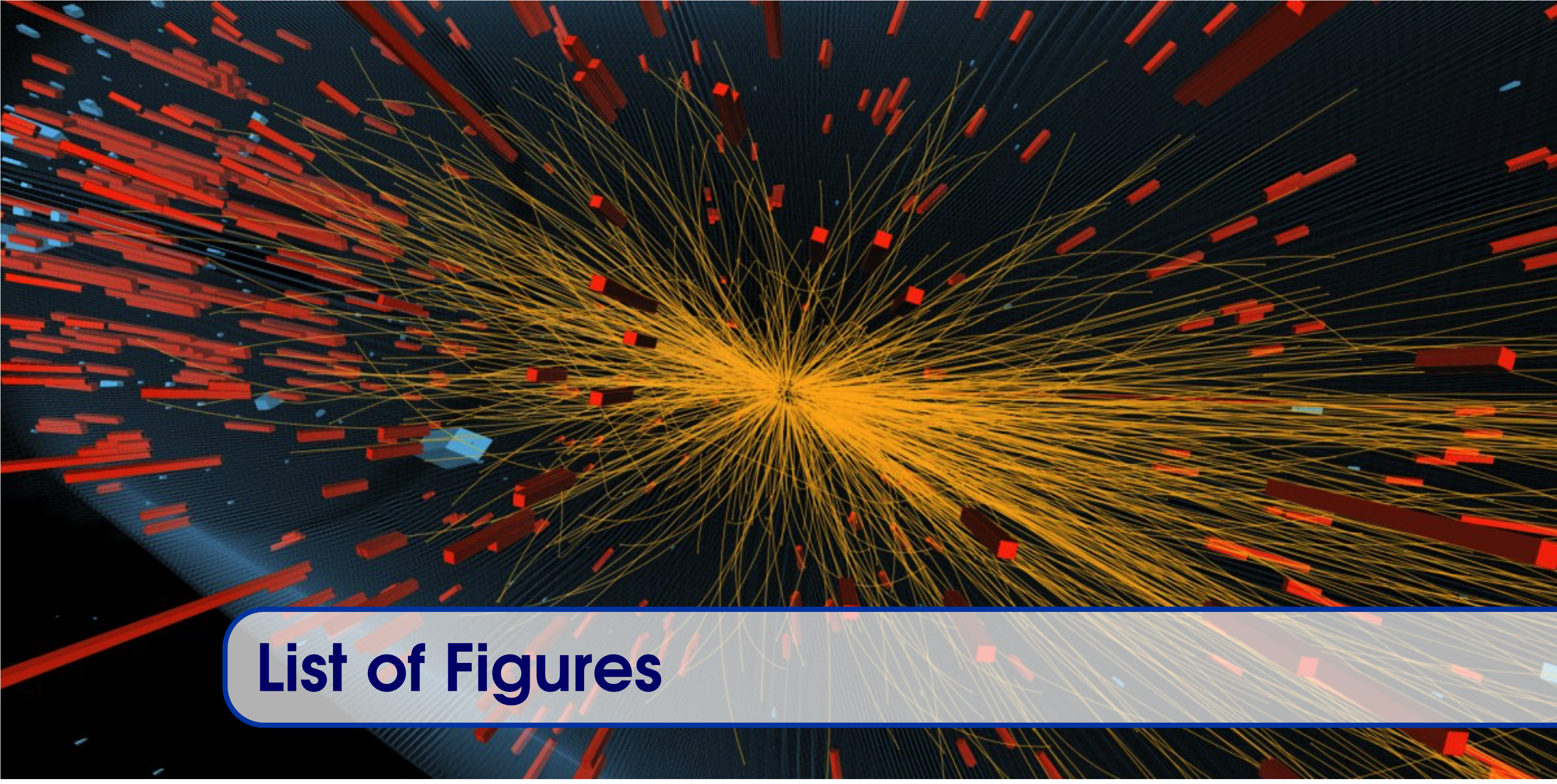
At the receiving end of the network, the CTRs recover the clock [3] and decode the signals in order to produce equipment-specific events, the so-called Local Timing events (LTIM). The events can be electrical pulses or software interrupts. The CTRs are very flexible and can produce a wide variety of local timing, from a simple repetition of a central timing event, to a complex scheme to produce bursts of pulses, synchronised with the revolution frequency of the accelerator. The most common configuration is the repetition of a central timing event after resynchronisation with an accelerator clock. This scheme performs a shift from an absolute time domain to beam-related time domain. CTRs are available in all of the hardware formats (see chapter 5) supported by BE-CO.

Distributed timing is integrated into FESA via the timing event source, which relies upon the timdt library. Therefore a FESA developer is able to use a timing event as a logical event, which triggers RT actions (see section 11.2). In addition, the timdt library provides access to all of the contextual information related to an event and is typically used to de-multiplex settings and acquisition data (see chapter 4). There is a need for high-level applications to retrieve information about events such as their timestamp and other contextual data. The XTIM FESA class provides this functionality. Every XTIM device shadows a specific central timing event and makes the required information available via RDA3.

When an LTIM is configured to produce an electrical pulse on a CTR output, the pulse is transported to the receiving equipment via cables. When a pulse has to be fanned out or the distance is too great for the Transistor-Transistor Logic (TTL) levels, pulse repeaters are placed in the distribution chain to propagate the signal. For longer distances, we rely on a CERN-made electrical standard called "Blocking". The pulse-repeaters are VME modules that can be placed in chassis with or without CPUs.

In 2019, more than 7000 LTIMs are used operationally, to orchestrate the vast majority of accelerator equipment. With such a critical mission, the timing system needs to be closely monitored in order to detect problems and help with troubleshooting. Recently, this monitoring has been transferred to the BE-CO solution, COSMOS, as described in section 19.1.

IX Extras



List of Figures

Glossary

.NET a software framework developed by Microsoft that runs primarily on Microsoft Windows. 87, 151

AcqCore shortened version of Acquisition Core. 98–100

ActiveMQ an open source message broker written in Java together with a full Java Message Service (JMS) client from Apache. 89

ACW A CERN-made framework for web development. 120

AD The Anti-proton Decelerator at CERN. 24

Angular a TypeScript-based open-source front-end web application framework led by Google. 126

Angular JS a JavaScript-based open-source front-end web application framework maintained by Google. 125, 146

ANSI/VITA VITA is an incorporated, non-profit organization of vendors and users having a common market interest in real-time, modular embedded computing systems. The VITA Standards Organization (VSO), the standards development arm of VITA, is accredited as an American National Standards Institute (ANSI) developer and a submitter of Industry Trade Agreements to the IEC. 38

Ansible open source software that automates software provisioning, configuration management, and application deployment. 59, 61

APEX Oracle Application Express (APEX) is a web-based software development environment that runs on an Oracle database. 146

API An Application Programming Interface (API) is a particular set of rules and specifications that a software program can follow to access and make use of the services and resources provided by another particular software program that implements that API. 28

ARM previously Advanced RISC Machine, originally Acorn RISC Machine, is a family of reduced instruction set computing architectures for computer processors, configured for various environments. 61, 80

Artifactory Proprietary binary repository manager. 135

AWAKE CERN experiment investigating the use of plasma wakefields driven by a proton bunch to accelerate charged particles. 104

AXI4 the fourth generation of the ARM Advanced Microcontroller Bus Architecture (AMBA) interface specification. 63

B&R a member of the ABB group, B&R Industrial Automation GmbH. is an Austrian manufacturer of automation technology. 45

Bash a Unix shell and command language developed as a free software for the GNU Project. 140

Bitstream A binary sequence used to transmit digital information such as the gateware of an FPGA. 39

Blocking CERN-made electrical standard based on 20-Volt pulses. 164

Bootstrap a HTML, CSS, and JS framework for developing responsive, mobile first projects on the web. 125, 126

BuildRoot a set of Makefiles and patches that simplifies and automates the process of building a complete and bootable Linux environment for an embedded system. 61

C a general-purpose, imperative computer programming language providing facilities for low-level manipulation. 66, 78, 140, 141, 180

C++ a general-purpose programming language providing facilities for low-level manipulation. 74, 75, 79, 80, 130, 134, 141, 147

CCC Main control room at CERN. 12

Collectd A Unix daemon that collects, transfers and stores performance data of computers and network equipmen. 139

Console Computer running high-level graphical applications. 12, 13, 17, 54, 58–60, 98, 112, 122, 167

CRC an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to raw data. 44

Cron The software utility Cron is a time-based job scheduler in Unix-like computer operating systems (Wikipedia). 59

Deployment The act of installing and configuring a version of software onto a target system. 58, 59, 74, 79, 89, 106

Eclipse an integrated development environment used in computer programming. 74, 79, 115, 133

Elasticsearch a search engine providing a distributed, multitenant-capable full-text search with an HTTP web interface. 144

ELENA a compact ring for cooling and further deceleration of 5.3 MeV antiprotons delivered by the CERN AD. 26

Encore a CERN-made tool to generate device drivers from VME hardware module description. 69, 70, 78, 142, 180

Etherbone an FPGA-core that connects Ethernet to internal on-chip Wishbone Buses permitting any core to talk to any other across Ethernet. `http://www.ohwr.org/projects/etherbone-core`. 80

EtherCAT an Ethernet-based fieldbus system, invented by Beckhoff Automation. 44

EthernetIP an industrial network protocol that adapts the Common Industrial Protocol (an industrial protocol for industrial automation applications) to standard Ethernet. 44

Fermilab Fermi National Accelerator Laboratory, located just outside Batavia, Illinois, near Chicago, is a United States Department of Energy national laboratory specializing in high-energy particle physics. 58

Font Awesome a suite of pictographic icons for scalable vector graphics on websites. 125

FuseSoC an open-source package manager and a set of build tools for HDL. 65

Git An open-source distributed Version Control System. 133

GitLab A web-based Git-repository manager. 133, 134

GNU An extensive and free collection of computer software, mostly licensed under GPL. 66

Gradle An open-source build automation system. 125, 136

Grafana An open-source, general purpose, web-based dashboard and graph composer. 140, 144

Groovy a Java-syntax-compatible object-oriented programming language for the Java platform from Apache. 118, 119

HAProxy open source software that provides a high availability load balancer and proxy server for TCP and HTTP-based applications. 146, 147

I/O The data or information that is passed into or out of a computer
The combination of devices, channels, and techniques controlling the transfer of information between a CPU and its peripherals. 15

Icinga2 An open-source computer system and network monitoring application. 139, 140

IEEE 1355 a data communications standard for Heterogeneous Interconnect (HIC). 176

IEEE 1588 also known as PTP is a protocol used to synchronize clocks throughout a computer network. 46

InfluxDB An open-source time series database developed by InfluxData. 139

IntelliJ a Java integrated development environment for developing computer software. 133

ioctl a system call for device-specific input/output operations and other operations which cannot be expressed by regular system calls. 67

ISOLDE The On-Line Isotope Mass Separator is a radioactive ion beam facility at CERN. 24, 100

IVI Instrument driver specification with aim of unifying hardware and software to achieve 'plug and play' interoperability for compatible instruments. 48

Java a general-purpose computer-programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. 12, 14, 51, 84, 87–91, 98, 101, 103, 105–107, 110, 115, 116, 118–122, 125, 126, 130, 132, 134, 141, 143, 144, 146, 159, 160, 163, 181

JavaFX a software platform for creating and delivering desktop applications. 120–122, 124, 126

JavaScript a high-level, interpreted programming language that is one of the core technologies of the World Wide Web. Often abbreviated to JS. 171, 176, 177, 181

Jenkins An open source automation server for continuous integration. 134

Jira issue tracking product developed by Atlassian. 55

JSON an open-standard file format that uses human-readable text to transmit data objects consisting of attribute–value pairs and array data types. 123

Kafka an open-source stream-processing software platform provided by the Apache Software Foundation. 144

Kernel the core of a computer's operating system, with complete control over everything in the system. 37, 58, 60, 61, 67–69

Kibana an open source data visualization plugin for Elasticsearch. 144

Knob A panel in an application for modifying a parameter. 129–131

Kotlin a statically typed, multi-platform general-purpose programming language, with type inference. 119

Kubernetes an open-source container-orchestration system for automating deployment, scaling and management of containerized applications. 107

LabVIEW a system-design platform and development environment for a visual programming language from National Instruments. LabVIEW stands for Laboratory Virtual Instrument Engineering Workbench. 35, 79

LEIR The Low Energy Ion Ring is part of the LHC ion chain at CERN. 25, 28, 89, 114, 119, 163

LHC The Large Hadron Collider is CERN's largest accelerator. 24

LINAC2 A Linear Accelerator at CERN accelerating protons up to 50 MeV. The LINAC2 was decommisioned in 2018. 25, 28

LINAC3 A Linear heavy ion Accelerator, part of the LHC ion chain at CERN. 25, 28

LINAC4 A Linear Accelerator at CERN accelerating H- up to 160 MeV.. 28, 104

Linux a family of free and open-source software operating systems based on the Linux kernel by Linus Torvalds. 58, 141

LM32 LatticeMico32 is a 32-bit microprocessor soft core from Lattice Semiconductor optimized for field-programmable gate arrays (FPGAs). 65

Logstash an open source, server-side data processing pipeline that ingests data from a multitude of sources simultaneously. 144

LynxOS RTOS is a Unix-like real-time operating system from Lynx Software Technologies (formerly "LynuxWorks"). 14, 34, 60

Macrocycle Fixed-length WorldFIP cycle. 42, 44, 45

Makerule An algorithm used in InCA/LSA. 95, 100

MBean a managed Java object, similar to a JavaBeans component, that follows the design patterns set forth in the JMX specification. 141

Middleware software that enables communication and management of data in distributed applications. 18, 85–87, 89–91

MockTurtle an HDL core of a generic Control System node, based on a deterministic multicore CPU architecture. 43, 58, 64–66

Modbus a serial communications protocol for use with programmable logic controllers (PLCs) and industrial electronic devices. 79

MPEG-4 a method of defining compression of audio and visual (AV) digital data. 123

Multiplexing The settings have different values at different times depending on the beam being produced. 23, 26, 28, 72, 85, 90, 100

Netbeans an integrated development environment for Java. 101

nTOF Studies neutron-nucleus interactions for neutron energies ranging from a few meV to several GeV. 24

Openwire a binary protocol designed for working with message-oriented middleware. It is the native wire format of Apache's ActiveMQ. 89

Operating System system software that manages computer hardware and software resources and provides common services for computer programs. E.g. Linux or Microsoft Windows. 58

OSI model The Open Systems Interconnection model (OSI model) is a conceptual model that characterises and standardises the communication functions of a telecommunication or computing system without regard to its underlying internal structure and technology. 44

Parameter Smallest controllable element in the Control System. 17–22, 69, 90–92, 94–96, 98–101, 106, 107, 109, 117, 121, 122, 124, 128–131, 151

PCI-SIG Peripheral Component Interconnect Special Interest Group is an electronics industry consortium responsible for specifying the Peripheral Component Interconnect, PCI-X, and PCI Express computer buses. 34

PICMG Consortium of companies who collaboratively develop open standards for high performance telecommunications and industrial computing applications. 14

Pnuts a dynamic scripting language for the Java platform. 115, 116

POSIX a family of standards specified by the IEEE Computer Society for maintaining compatibility between operating systems. 60

POWERLINK a deterministic, real-time, open protocol for standard Ethernet. 44, 45

Profinet an industry technical standard for data communication over Industrial Ethernet. 44

Prometheus A free software ecosystem for monitoring and alerting. 140

PS complex The accelerators below SPS including facilities downstream of the PSB and the PS such as ISOLDE, AD, etc. 130, 158

PVSS An industrial Control System developed by Siemens. 79

PXE specification describing a standardised client-server environment that boots a software assembly, retrieved from a network, on PXE-enabled clients. 60

PyDM Python Display Manager, a Rapid Application Development (RAD) tool developed by SLAC. 127

PyJAPC A Python binding for JAPC. 127

PyPI A repository of software for the Python programming language. 136

PyQt A Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. 127, 143

Python An interpreted, high-level, general-purpose programming language. 69, 106, 120, 126, 127, 134, 136, 140, 142, 146, 147, 175

PyUAL A Python binding for UAL. 142

Qt A cross-platform application framework and widget toolkit for creating desktop and embedded graphical user interfaces. 80, 122, 124, 175

Qt Designer A graphical tool that lets you build Qt GUIs. 127

RAMSES system used to monitor radiation at CERN. 103

Reference value Also known as setting. Desired value of the parameter being controlled. 99, 128

REST REST is an architecture style for designing networked applications and is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls), in particular RMI in our case. 147

RS232 standard for serial communication transmission of data, originally introduced in 1960. 67

RS485 standard for serial multi-drop communication transmission of data. 163

S7 a Siemens proprietary protocol that runs between programmable logic controllers (PLCs) of the Siemens S7-300/400 family. 79

Schneider a French industrial manufacturing company. 79

Setpoint The desired or target value for an essential variable, or process value of a system. 94

Setting see reference value. 10, 17–19, 21–23, 72, 73, 85, 90, 94–102, 128, 130, 131, 158, 159

Siemens One of the largest industrial manufacturing companies in Europe, headquartered in Munich, Germany. 44, 79, 175–177

SLEquip Obsolete Front-End Computer software framework. 75

SonicMQ a Java Message Service (JMS) broker. 89

SpaceWire a spacecraft communication network based in part on the IEEE 1355 standard of communications. 44

Spring an application framework and inversion of control container for the Java platform. 84, 125, 126

SPS The Super Proton Synchrotron (SPS) is the second-largest machine in CERN's accelerator complex, measuring nearly 7 kilometres in circumference. 25, 26, 28, 94, 95, 97, 98, 100, 103, 104, 108, 109, 114, 160, 163, 164

Stomp Simple Text Oriented Message Protocol, formerly known as TTMP, is a simple text-based protocol, designed for working with message-oriented middleware. 89

SVN Version-control software. 79

Swing a GUI widget toolkit for Java, part of Oracle's Java Foundation Classes providing a graphical user interface for Java programs. 12, 101, 120–124, 126, 130–132, 143, 160, 163

TANGO A free, open source, device-oriented controls toolkit for controlling any kind of hardware or software and building SCADA systems.. 127

Taurus A python framework for control and data acquisition CLIs and GUIs in scientific/industrial environments. 127

TIMBER Generic application for extracting and visualising logged data. 102, 104

timdt Low-level timing library used in FEC software. 164

Trim A small change to a parameter's setting value. 18, 90, 94–96, 122, 130

TT40 a transfer line between the SPS and the LHC. 104

TypeScript an open-source programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript, and adds optional static typing to the language. 125

U Unit of measurement for electronics components (3U, 6U, 45U). 31, 51, 52

Velocity a Java-based template engine that provides a template language to reference objects defined in Java code. 118, 119

Verilog a hardware description language (HDL) used to model electronic systems, standardized as IEEE 1364. 63

VHDL a hardware description language (HDL) used to model electronic systems. The V in VHDL stands for VHSIC (Very High Speed Integrated Circuit). 38, 44, 63

VPX The other name of the VITA 46 standard. 33

Webpack an open-source JavaScript module bundler. 125

White Rabbit Augmented Ethernet Network Protocol. 38, 44, 47, 61, 64, 66, 160, 183

WinCC OA An industrial Control System developed by Siemens. 50, 79, 102

Wishbone Bus an open source hardware computer bus intended to let the parts of an integrated circuit communicate with each other. 63–65, 78, 80, 172

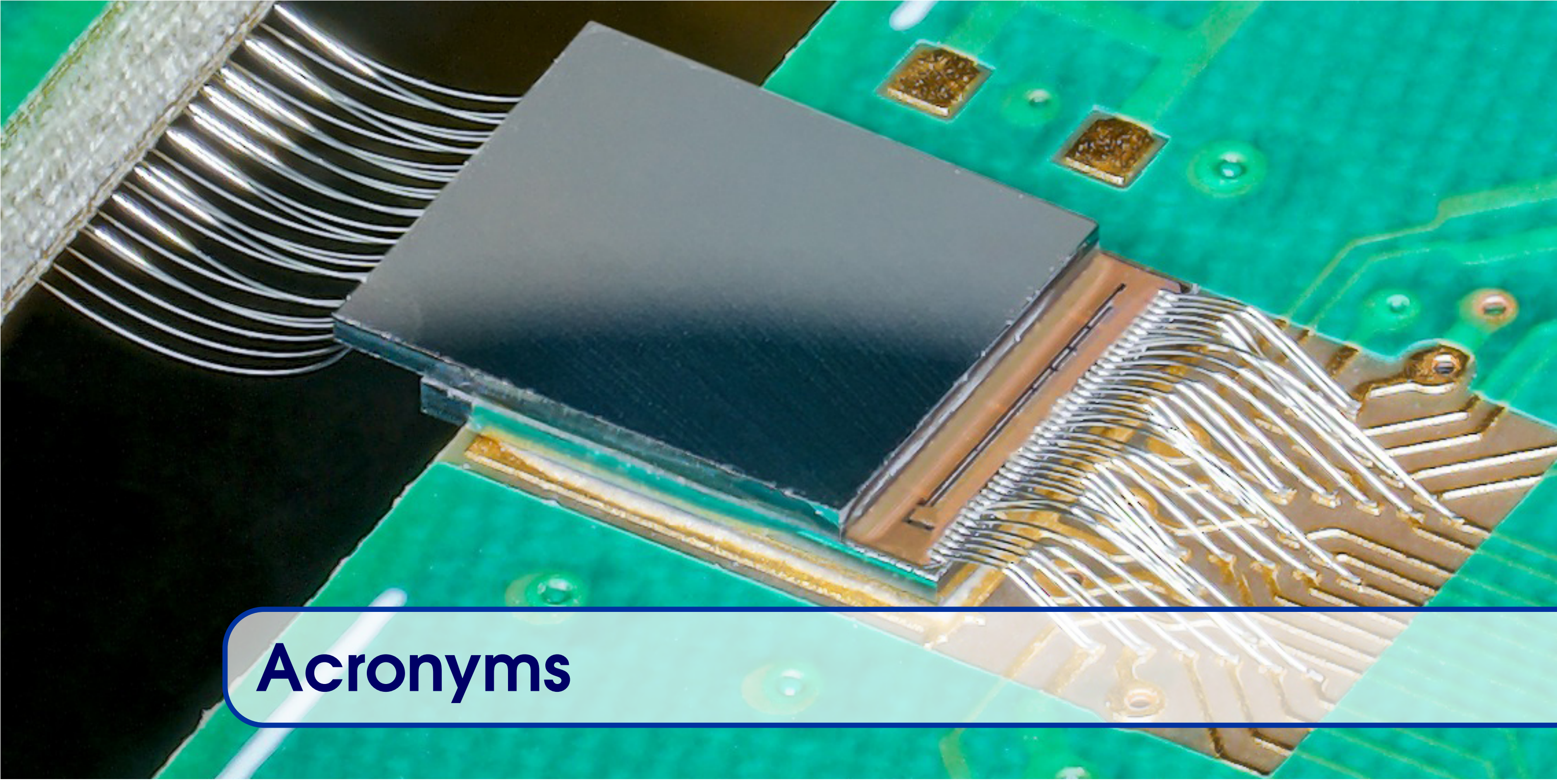
WorkingSet A table of devices. 98, 128–130

X-motif a widget toolkit for building graphical user interfaces under the X Window System on Unix. 122, 130, 132

XTIM FESA class to expose central timing distributed information on RDA3. 164

YAML a human-readable data serialization language, commonly used for configuration files. 78

zeroMQ a high-performance asynchronous messaging library, aimed at use in distributed or concurrent applications. 86



Acronyms

μRV micro RISC 5. 65

ABI Application Binary Interface. 68
ABS Assembly Breakdown Structure. 151
ACW Accsoft Commons Web. 120, 125, 126, 146, 152, 154, 155, *Glossary:* ACW
AD Anti-proton Decelerator. 24, 26, 73, 98, *Glossary:* AD
ADC Analogue-to-Digital Converter. 37–39, 49, 63
AFT Accelerator Fault Tracking. 153–155
AMC Advanced Mezzanine Card. 36, 37
AMT Active Management Technology. 54, 82, 83
API Application Programming Interface. 28, 48, 49, 67–70, 78, 86, 89–91, 101, 102, 104, 110, 118, 134, 141, 142, 145–147, 154, 155, 181, *Glossary:* API
ASM Accelerator Schedule Management. 153–155
ATCA Advanced Telecommunications Computing Architecture. 36
AWAKE Advanced Proton Driven Plasma Wakefield Acceleration Experiment. 104, *Glossary:* AWAKE
AWG Availability Working Group. 153, 154
AXI4 Advanced eXtensible Interface 4. 63, 78, *Glossary:* AXI4

BC Bus Controller. 41
BCD Beam Coordination Diagram. 26, 162, 163
BE Beams Department. 9, 33
BE-BI Beam Instrumentation group. 9, 116
BE-CO Controls group. 3, 9, 30, 32, 34–41, 54–58, 64, 66, 68, 69, 73, 77, 79, 88, 101, 120, 124, 125, 127, 133, 136, 152–154, 158, 159, 164, 165
BE-OP Operations group. 9
BE-RF Radio Frequency group. 33, 83

BIS Beam Interlock System. 116
BLM Beam Loss Monitor. 89, 108, 109
BMA Block Memory Access. 33

C2MON CERN Control and Monitoring Platform. 113
CALS CERN-wide Accelerator Logging Service. 102, 104, 105
CAS CERN Alarm System. 113
CBNG Common Build Next Generation. 134–136
CC7 CERN CentOS 7. 60
CCC CERN Control Centre. 12, 54, *Glossary:* CCC
CCDA Controls Configuration Data Access. 145–147
CCDB Controls Configuration Database. 56, 61, 69, 75, 78, 83, 106, 112, 131, 143, 145, 146, 160
CCDE Controls Configuration Data Editor. 145–147
CCM Common Console Manager. 131, 132, 146
CCR Controls Computer Room. 12, 54, 82
CCS Controls Configuration Service. 107, 145, 146, 154
CentOS Community Enterprise Operating System. 54, 58, 60, 61, 80, 179
CESAR CERN Experimental Area SoftwAre Renovation. 50, 100–102
CI Continuous Integration. 134
CLI Command Line Interface. 142
CMMS Computerised Maintenance Management System. 55
CMW Controls Middleware. 84–90, 104–106, 126, 134, 143, 144, 163
CMX C++ Management Extension. 141
CompactPCI Compact Peripheral Component Interconnect. 34, 158, 159
COMRAD COntrols Multi-purpose Rapid Application Development. 126, 127
CORBA Common Object Request Broker Architecture. 86, 109, 110
COSMOS Controls Open-Source Montoring System. 139–141, 165
COTS Commercial-Off-The-Shelf. 32, 34, 38–40, 45
CPU Central Processing Unit. 12, 14, 30, 32–35, 43, 50, 51, 57, 61, 65, 66, 80, 82, 86, 165
CRC Cyclic Redundancy Check. 44, *Glossary:* CRC
CRM Cluster Resource Manager. 59
CSS Cascading Style Sheets. 125
CT Central Timing. 162
CTB Controls Testbed. 134
CTIM Central Timing events. 27
CTR Central Timing Receiver. 162, 164

DB Database. 83, 84, 89
DDS Direct Digital Synthesizer. 47
DHCP Dynamic Host Configuration Protocol. 60
DIAMON Diagnostic and Monitoring System. 139, 141
DIP Data Interchange Protocol. 86
DMA Direct Memory Access. 33, 70
DSL Domain Specific Language. 118, 125

ECC Error-Correcting Code. 51

EDGE Encore Driver GEnerator. 69, 70, 78, 142

EDMS Electronic Document Management System. 151

EN Engineering Department. 9, 55, 56

EN-ACE Accelerator Coordination and Engineering group. 55

EN-SMM Survey Mechatronics Measurements group. 9, 37, 39

EP Experimental Physics Department. 136

ESA European Space Agency. 44

FAAS Function As A Service. 107

FCC Future Circular Collider. 152

FDF Fixed Display Framework. 123

FEC Front-End Computer. 14, 21, 23, 30, 43, 45, 60, 69, 72, 73, 75, 80, 83, 84, 88, 94, 96, 98, 99, 108, 109, 114, 159, 163, 176

FESA Front-End Software Architecture. 28, 37, 66, 67, 69, 73–76, 78–80, 83, 84, 90, 91, 101, 102, 105, 106, 122, 130, 134, 141–144, 146, 155, 159, 163, 164, 177

FGC Function Generator Controller. 75, 84, 102, 108, 109, 144

FIFO First In First Out. 66

FIPdiag Diagnostic Agent for WorldFIP. 42

FMC FPGA Mezzanine Card. 38–40, 43, 45, 47, 49, 63, 183

FOSS Free Open Source Software. 59

FP Functional Position. 150, 151

FPGA Field-Programmable Gate Array. 34, 38, 39, 43, 45, 63–65, 69, 77, 79, 80, 180

GCC GNU C Compiler. 66

GCS Group Communication System. 59

GDB GNU Debugger. 66

GM General Module. 75, 102, 147

GMT General Machine Timing. 27, 28, 42, 46, 47, 162–164

GNU GNU's Not Unix. 66, 180, *Glossary:* GNU

GPN General Purpose Network. 82, 89, 141

GPS Global Positioning System. 163

GUI Graphical User Interface. 12, 14, 17, 18, 20, 61, 79, 84, 88, 92, 101, 102, 115, 118–122, 127, 130, 131, 139, 141, 146, 147, 160, 163, 175

HBA Host Bus Adapter. 51, 53

HDD Hard Disk Drive. 50, 51

HDL Hardware Description Language. 63–65, 69, 77, 78

HL-LHC High-Luminosity Large Hadron Collider. 44, 103, 152

HMI Human Machine Interface. 120

HPC High Pin Count. 38

I/O Input/Output. 15, 35, 37–39, 67, 75, 80, *Glossary:* I/O

I2C Inter-Integrated Circuit. 38

IDE Integrated Development Environment. 74, 115, 119, 133, 134

IEPLC Ethernet Interconnection for Programmable Logic Controllers. 79

INB Installation Nucleaire de Base (Basic Nuclear Facility). 102

InCA Injector Control Architecture. 50, 89, 90, 94, 98, 99, 106, 121–123, 128, 129

InforEAM Enterprise Asset Management System. 55, 151

IPMI Intelligent Platform Management Interface. 37, 52, 54, 83, 139
IQC Injection Quality Check. 108
ISIC Individual Software Interlock Channel. 117, 118
IT Information Technology Department. 52, 54, 56, 58–60, 83, 144
IVI Interchangeable Virtual Instrumentation. 48, *Glossary:* IVI

JAPC Java API for Parameter Control. 89–92, 105, 106, 109, 118, 121–124, 175, 181
JBOD Just a Bunch Of Disks. 50, 52, 53
JDBC Java Database Connectivity. 103
JMON JAPC monitoring. 92, 106, 119, 124
JMS Java Message Service. 14, 87–89, 101, 105, 110, 112, 160
JMX Java Management Extensions. 89, 141
JSON JavaScript Object Notation. 123, *Glossary:* JSON

L865 32-bit SLC5. 60
L866 64-bit SLC6. 60
LAN Local Area Network. 41, 51
LASER LHC Alarms Service. 111–113, 154
LCG LHC Computing Grid. 136
LDB Logging Database. 104
LEMON LHC Era Monitoring. 139
LGPL Lesser General Public Licence. 66, 86
LHC Large Hadron Collider. 24–26, 28, 31, 35, 41, 42, 44, 45, 47, 73, 83, 86, 89, 94, 97, 98, 100, 102–104, 107, 108, 111, 114–117, 120, 123, 128, 136, 139, 150, 152–156, 158, 160, 164, 181, *Glossary:* LHC
LIC LHC Injector Chain. 25, 26
LIST LHC Instability Study Triggers. 47, 48
LIU LHC Injector Upgrade. 152
LPC Low Pin Count. 38
LS1 Long Shutdown 1. 134
LS2 Long Shutdown 2. 34, 44, 57, 60, 86, 105, 144
LSA LHC Software Architecture. 94, 95, 97, 98, 100–103, 130, 131
LSIC Logical Software Interlock Channel. 117
LTIM Local Timing events. 27, 164, 165
LVM Logical Volume Manager. 53
LXI LAN eXtensions for Instrumentation. 48, 49

MCC Machine Controls Coordinator. 55
MCH MicroTCA Carrier Hub. 36, 37
MD Machine Development. 19, 114, 155, 156
MDB Measurement Database. 104
MPL Mozilla Public Licence. 86
MRG Messaging Real-time and Grid. 60
MTBF Mean Time Between Failure. 54
mTCA Micro Telecommunications Computing Architecture. 36, 37, 159
MTF Manufacturing and Test Folder. *Glossary:* MTF
MTT Multi-Tasking Timing. 163, 164

NFS Network File System. 50, 53, 61, 110
NIC Network Interface Controller. 45
NPM Node.js Package Manager. 125
nTOF Neutron Time-Of-Flight facility. 24, 27, *Glossary:* nTOF
NXCALS Next CERN Accelerator Logging Service. 105, 110, 111, 144

OASIS Open Analogue Signal Information System. 34, 47, 48, 88, 106, 158–160
OS Operating System. 58, 60, 139, *Glossary:* Operating System

PCB Printed Circuit Board. 35, 39, 63
PCI Peripheral Component Interconnect. 14, 30, 32, 34–37, 39–41, 51, 64, 68, 70, 82, 142, 159, 182
PCIe PCI Express. 34–39, 43, 45, 51, 63, 64, 68, 70, 82, 159, 183
PCP Priority Code Point. 47
PICMG PCI Industrial Computer Manufacturers Group. 14, 32, 36, *Glossary:* PICMG
PID Process IDentification number. 141
PLC Programmable Logic Controller. 15, 45, 79, 163
PM Post-Mortem. 108–110
PMA Post-Mortem Analysis. 109, 110
PMFE Post-Mortem Front-End. 108, 109
PPM Pulse-to-Pulse Modulation. 25, 122
PPS Pulse-Per-Second. 163
PS Proton Synchrotron. 24–26, 28, 89, 98, 100, 119
PSB PS Booster. 24–26, 28, 112, 119
PTP Precise-Time-Protocol. 46, 64, 173
PXI PCI eXtension for Instrumentation. 35, 37, 182
PXIe PXI Express. 37–39, 79, 82, 183
PyPI Python Package Index. 136, *Glossary:* PyPI

RAD Rapid Application Development. 175
RAID Redundant Array of Inexpensive Disks. 51–53, 109
RAM Random Access Memory. 30, 33, 34, 38, 60, 61, 78, 98
RBAC Role-Based Access Control. 83–85, 102, 125
RCP Rich Client Platform. 79
RCS Revision Control System. 133
RDA Remote Device Access. 84–90, 106, 108, 143, 164, 177
REST REpresentational State Transfer. 147, 155, *Glossary:* REST
RHEL Red Hat Enterprise Linux. 58
RISC Reduced Instruction Set Computer. 65, 178
RMI Remote Method Invocation. 14, 84, 88, 101, 147, 160, 163
RT Real-Time. 18, 20, 77, 78, 141, 159, 164
RTI Remote Terminal Interface. 41
RTM Rear-Transition Module. 31, 37, 39, 164

SAS Serial Attached SCSI. 53
SBC Single Board Computer. 30, 33, 34, 82, 98
SCADA Supervisory Control and Data Acquisition. 79
SCSI Small Computer System Interface. 53, 182

SCSS Sassy Cascading Style Sheets. 125
SFP Small Form-factor Pluggable. 38
SHB System Host Board. 32
SIG Special Interest Group. 35
SILECS Software Infrastructure for Low-Level Equipment ControllerS. 77, 79, 80, 163
SIS Software Interlock System. 89, 110, 116–119
SLAC Stanford Linear Accelerator Center. 127, 175
SLC Scientific Linux CERN. 54, 60, 61, 80, 181
SMB-SMS Service Management and Support group. 55
SNMP Simple Network Management Protocol. 37, 47, 83, 139
SNOW Service Now. 55
SPEC Simple PCIe FMC Carrier. 39, 43, 63
SPEXI Simple PXIe FMC Carrier. 39
SQL Structured Query Language. *Glossary:* SQL
SSD Solid-State Disk. 50, 52
SSH Secure Shell. 59
SVEC Simple VME FMC Carrier. 39, 48, 49
SVN Apache Subversion. 79, 133, *Glossary:* SVN

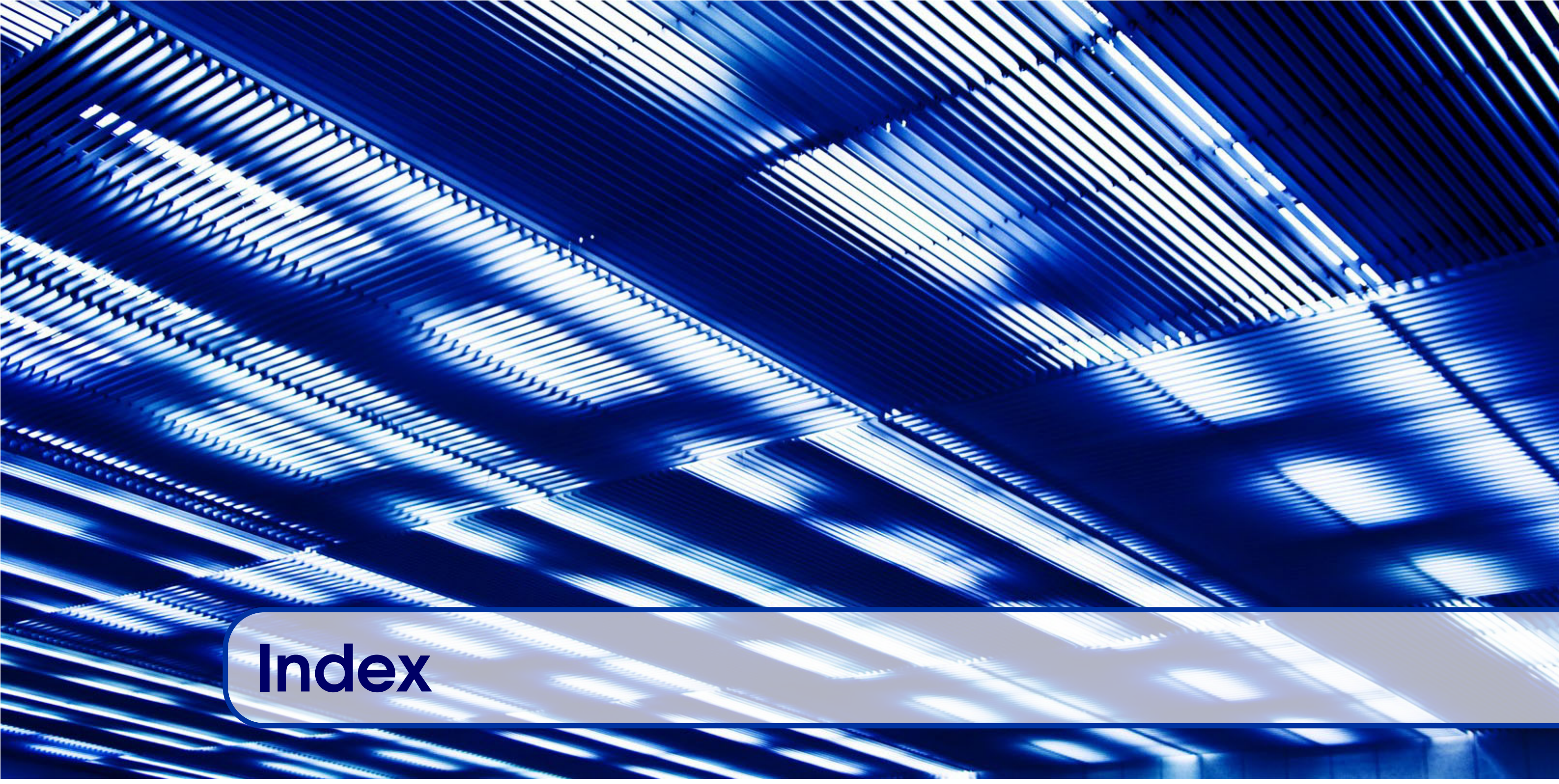
TDC Time-to-Digital Converter. 37, 39, 47–49
TE Technology Department. 9
TE-EPC Electrical Power Converters group. 9, 75
TFTP Trivial File Transfer Protocol. 60
TGM TeleGraM. 28, 89, 122
TI Technical Infrastructure. 111, 113
TIM Technical Infrastructure Monitoring. 103
TN Technical Network. 82, 83, 89, 141, 144
TTL Transistor-Transistor Logic. 164

UAL Unified Access Library. 142
UCAP Unified Controls Acquisition and Processing Framework. 106, 107, 119, 124
UFO Unidentified Falling Object. 153
UPS Uninterruptable Power Supply. 54
UTC Coordinated Universal Time. 46, 66

VCS Version Control System. 115, 133, 134
VME Versa Module Europa. 14, 30–36, 38, 39, 41, 49, 63, 64, 68–70, 82, 142, 159, 164, 183
VPX VME PCI eXtension. 33, *Glossary:* VPX
VXI VME eXtension for Instrumentation. 34, 35, 158, 159
VXS VME Switched Serial. 33

wbgen Wishbone Generator. 64, 65, 78
WorldFIP World Factory Instrumentation Protocol. 31, 39, 41–45, 180
WRTD White Rabbit Trigger Distribution. 44, 48, 160

XML Extensible Markup Language. 79, 112, 118, 119, 123, 124, 143
XPOC eXternal Post-Operation Check. 108



Index

Bibliography

[1] P. Alvarez, J. Lewis, and J. Serrano. "The LHC Central Timing Hardware Implementation". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'07), 15-19 October 2007* (Knoxville, Tennessee, USA), pages 400–402. URL: `http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/WPPB02.PDF` (cited on page 163).

[2] P. Alvarez et al. "FPGA Mezzanine Cards for CERN's Accelerator Control System". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09), 12-16 October 2009* (Kobe, Japan), pages 376–378. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/web002.pdf` (cited on page 38).

[3] P. Alvarez et al. "PLL Usage in the General Machine Timing System for the LHC". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'03), 13-17 October 2003* (Gyeongju, Korea), pages 116–118. URL: `http://accelconf.web.cern.ch/AccelConf/ica03/PAPERS/MP532.PDF` (cited on page 164).

[4] A. Apollonio et al. "LHC Accelerator Fault Tracker - First Experience". In: *Proc. of International Particle Accelerator Conference (IPAC'16), May 8-13, 2016* (Busan, Korea), pages 1190–1192. URL: `http://jacow.org/ipac2016/papers/tupmb040.pdf` (cited on page 153).

[5] M. Arruat et al. "Front-End Software Architecture". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'07), 15-19 October 2007* (Knoxville, Tennessee, USA), pages 310–312. URL: `http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/WOPA04.PDF` (cited on page 73).

[6] V. Baggiolini et al. "A Sequencer for the LHC Era". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09),*

12-16 October 2009 (Kobe, Japan), pages 670–672. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/thc003.pdf` (cited on page 114).

[7] V. Baggiolini et al. "The CESAR Project - Using J2EE for Accelerator Controls". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'03), 13-17 October 2003* (Gyeongju, Korea), pages 269–271. URL: `http://accelconf.web.cern.ch/AccelConf/ica03/PAPERS/TU512.PDF` (cited on page 101).

[8] E. Van Der Bij et al. "Open Hardware for CERN's Accelerator Control Systems". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), 10-14 October 2011* (Grenoble, France), pages 554–557. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/tubault04.pdf` (cited on page 40).

[9] R. Billen et al. "Accelerator Data Foundation: How It All Fits Together". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09), 12-16 October 2009* (Kobe, Japan), pages 61–65. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/tub001.pdf` (cited on page 145).

[10] A. Bland and S.T. Page. "Upgrades to the Infrastructure and Management of the Operator Workstations and Servers for Run 2 of the CERN Accelerator Complex". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), 17-23 October 2015* (Melbourne, Australia), pages 1158–1161. URL: `http://jacow.org/icalepcs2015/papers/thhd3o08.pdf` (cited on page 59).

[11] L. Burdzanowski and C. Roderick. "The Renovation of the CERN Controls Configuration Service". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), 17-23 October 2015* (Melbourne, Australia), pages 103–106. URL: `http://jacow.org/icalepcs2015/papers/mopgf006.pdf` (cited on page 146).

[12] L. Burdzanowski et al. "CERN Controls Configuration Service - a Challenge in Usability". In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17), 8-13 October 2017* (Barcelona, Spain), pages 159–165. URL: `http://jacow.org/icalepcs2017/papers/tubpl01.pdf` (cited on page 146).

[13] F. Calderini et al. "Moving Towards a Common Alarm Service for the LHC Era". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'03), 13-17 October 2003* (Gyeongju, Korea), pages 580–582. URL: `http://accelconf.web.cern.ch/AccelConf/ica03/PAPERS/TH512.PDF` (cited on page 113).

[14] M. Cattin et al. "CERN's FMC KIT". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), 06-11 October 2013* (San Francisco, CA, USA), pages 1020–1023. URL: `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/wecocb01.pdf` (cited on pages 38, 40).

[15] J. D. Gonzalez Cobas et al. "Free and Open Source Software at CERN: Integration of Drivers in the Linux Kernel". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), 10-14 October*

2011 (Grenoble, France), pages 1248–1251. URL: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/thchmust04.pdf (cited on page 68).

[16] J. Cuperus, R. Billen, and M. Lelaizant. “The Configuration Database for the CERN Accelerator Control System”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’03), 13-17 October 2003* (Gyeongju, Korea), pages 309–311. URL: http://accelconf.web.cern.ch/AccelConf/ica03/PAPERS/WE114.PDF (cited on page 146).

[17] G. Daniluk et al. “Solving Vendor Lock-in in VME Single Board Computers through Open-sourcing of the PCIe-VME64x Bridge”. In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS’17), 8-13 October 2017* (Barcelona, Spain), pages 131–136. URL: http://jacow.org/icalepcs2017/papers/tuapl03.pdf (cited on page 34).

[18] G. Daniluk and E. Gousiou. “Plans at CERN for Electronics and Communication in the Distributed I/O Tier”. In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS’17), 8-13 October 2017* (Barcelona, Spain), pages 1552–1556. URL: http://jacow.org/icalepcs2017/papers/thpha071.pdf (cited on page 15).

[19] S. Deghaye et al. “CERN Proton Synchrotron Complex High-Level Controls Renovation”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’09), 12-16 October 2009* (Kobe, Japan), pages 638–640. URL: http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/tha005.pdf (cited on page 94).

[20] S. Deghaye et al. “Hardware Abstraction Layer in OASIS”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’05), 10-14 October 2005* (Geneva, Switzerland), pages 1–5. URL: http://accelconf.web.cern.ch/AccelConf/ica05/proceedings/pdf/P1_090.pdf (cited on page 159).

[21] S. Deghaye et al. “OASIS: a New System to Acquire and Display the Analog Signals for LHC”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’03), 13-17 October 2003* (Gyeongju, Korea), pages 359–361. URL: http://accelconf.web.cern.ch/AccelConf/ica03/PAPERS/WP502.PDF (cited on page 158).

[22] S. Deghaye et al. “OASIS: Status Report”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’05), 10-14 October 2005* (Geneva, Switzerland), pages 1–6. URL: http://accelconf.web.cern.ch/AccelConf/ica05/proceedings/pdf/O4_007.pdf (cited on page 158).

[23] L. N. Drosdal et al. “Automatic Injection Quality Checks for the LHC”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’11), 10-14 October 2011* (Grenoble, France), pages 1077–1080. URL: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/wepmu011.pdf (cited on page 108).

[24] A. Dworak and J.C. Bau. “Decoupling CERN Accelerators”. In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS’17), 8-13 October 2017* (Barcelona, Spain), pages 608–611. URL: http://jacow.org/icalepcs2017/papers/tupha084.pdf (cited on page 26).

[25] F. Ehm et al. “CMX - a Generic Solution to Expose Monitoring Metrics in C and C++ Applications”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’13), 06-11 October 2013* (San Francisco, CA, USA), pages 1118–1121. URL: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/thppc014.pdf (cited on page 141).

[26] E. Fejes. “Adapting Gradle for the CERN Accelerator Controls System”. In: *Proc. of Gradle Summit 2017* (Palo Alto, CA, USA). URL: https://www.gradlesummit.com/topics/adapting_gradle_for_the_cern_accelerator_control_system (cited on page 136).

[27] E. Fortescue-Beck, R. Billen, and P. Gomes. “The LHC Functional Layout Database as Foundation of the Controls System”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’11), 10-14 October 2011* (Grenoble, France), pages 147–150. URL: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/mopkn024.pdf (cited on page 150).

[28] M. Gabriel and R. Gorbonosov. “Disruptor - Using High Performance, Low Latency Technology in the CERN Control System”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’15), 17-23 October 2015* (Melbourne, Australia), pages 606–609. URL: http://jacow.org/icalepcs2015/papers/web3o03.pdf (cited on page 101).

[29] L. Gallerani. “Large Graph Visualization of Millions of Connections in the CERN Control System Network Traffic: Analysis and Design of Routing and Firewall Rules with a New Approach”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’15), 17-23 October 2015* (Melbourne, Australia), pages 799–801. URL: http://jacow.org/icalepcs2015/papers/wepgf045.pdf (cited on page 82).

[30] J.C. Garnier et al. “Smooth Migration of CERN Post Mortem Service to a Horizontally Scalable Service”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’15), 17-23 October 2015* (Melbourne, Australia), pages 806–809. URL: http://jacow.org/icalepcs2015/papers/wepgf047.pdf (cited on page 111).

[31] R. Gorbonosov et al. “Plug-in Based Analysis Framework for LHC Post-Mortem Analysis”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’13), 06-11 October 2013* (San Francisco, CA, USA), pages 446–448. URL: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/moppc143.pdf (cited on page 109).

[32] F. Hoguin and S. Deghaye. “Solving the Synchronization Problem in Multi-Core Embedded Real-Time Systems”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’15), 17-23 October 2015* (Melbourne, Australia), pages 942–946. URL: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2015/papers/wepgf102.pdf (cited on page 74).

[33] D. Jacquet. “Breaking the wall between operational and expert tools”. In: *Proc. of the 2016 Evian workshop on LHC beam operation, 13-15 December 2016* (Evian, France), pages 157–160. URL: http://cds.cern.ch/record/2289585/files/Evian_CERN-ACC-2017-094.pdf (cited on page 127).

[34] K.Sigerud et al. “First Operational Experience With LASER”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’05), 10-14 October 2005* (Geneva, Switzerland). URL: `https://edms.cern.ch/ui/file/1825056/1/ICALEPCS-2005.pdf` (cited on page 113).

[35] Q. King. “Status of the LHC Power Converter Controls”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’09), 12-16 October 2009* (Kobe, Japan), pages 4–6. URL: `https://accelconf.web.cern.ch/accelconf/icalepcs2009/papers/mob003.pdf` (cited on page 75).

[36] G. Kruk, O. Da Silva Alves, and L. Molinari. “JavaFX Charts: Implementation of Missing Features”. In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS’17), 8-13 October 2017* (Barcelona, Spain), pages 866–868. URL: `http://jacow.org/icalepcs2017/papers/tupha186.pdf` (cited on page 121).

[37] G. Kruk and M. Peryt. “JDataViewer - Java-based Charting Library”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’09), 12-16 October 2009* (Kobe, Japan), pages 856–858. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/thp093.pdf` (cited on page 120).

[38] G. Kruk et al. “How to Successfully Renovate a Controls System? - Lessons Learned from the Renovation of the CERN Injectors’ Controls Software”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’13), 06-11 October 2013* (San Francisco, CA, USA), pages 43–46. URL: `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/mocobab05.pdf` (cited on page 94).

[39] G. Kruk et al. “LHC Software Architecture [LSA] – Evolution Toward LHC Beam Commissioning”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’07), 15-19 October 2007* (Knoxville, Tennessee, USA), pages 307–309. URL: `http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/WOPA03.PDF` (cited on page 94).

[40] J. Lewis et al. “The Evolution of the CERN SPS Timing System for the LHC Era”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’03), 13-17 October 2003* (Gyeongju, Korea), pages 125–127. URL: `http://accelconf.web.cern.ch/AccelConf/ica03/PAPERS/MP535.PDF` (cited on page 24).

[41] F. Locci and S. Magnoni. “IEPLC Framework, Automated Communication in a Heterogeneous Control System Environment”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’13), 06-11 October 2013* (San Francisco, CA, USA), pages 139–142. URL: `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/moppc031.pdf` (cited on page 79).

[42] N. Magnin et al. “External Post-Operational Checks for the LHC Beam Dumping System”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS’11), 10-14 October 2011* (Grenoble,

France), pages 1111–1114. URL: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/wepmu023.pdf (cited on page 108).

[43] S. Matthies et al. “FESA3 Integration in GSI for FAIR”. In: *Proc. of Personal Computers and Particle Accelerator Controls (PCaPAC), 14-17 October 2014* (Karlsruhe, Germany), pages 43–45. URL: http://accelconf.web.cern.ch/AccelConf/PCaPAC2014/papers/wpo006.pdf (cited on page 75).

[44] C. Pascual-Izarra et al. “Taurus big and small: from particle accelerators to desktop labs”. In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17), 8-13 October 2017* (Barcelona, Spain), pages 166–169. URL: http://jacow.org/icalepcs2017/papers/tubpl02.pdf (cited on page 127).

[45] G. Penacoba et al. “Design of an FPGA-based Radiation Tolerant Agent for WorldFIP Fieldbus”. In: *Proc. of International Particle Accelerator Conference (IPAC'11), September 4-9, 2011* (San Sebastian, Spain), pages 1780–1782. URL: http://accelconf.web.cern.ch/AccelConf/IPAC2011/papers/tups102.pdf (cited on page 43).

[46] M. Peryt et al. “Database and Interface Modifications: Change Management Without Affecting the Client”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), 10-14 October 2011* (Grenoble, France), pages 106–109. URL: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/MOPKN010.pdf (cited on page 146).

[47] T. Rendahl. “Pydm: a python alternative to edm”. In: *EPICS Collaboration Meeting, 16-23 September 2016* (Tennessee, USA). URL: http://conference.sns.gov/event/11/session/1/contribution/45/attachments/131/345/PYDM_.pdf (cited on page 127).

[48] C. Roderick. “CERN accelerator data logging and analysis”. In: *2013 IEEE Nuclear Science Symposium and Medical Imaging Conference (2013 NSS/MIC), 27 October - 2 November 2013* (Seoul, South Korea). URL: https://ieeexplore.ieee.org/document/6829573 (cited on page 103).

[49] C. Roderick and R. Billen. “The LHC Logging Service : Capturing, storing and using time-series data for the world's largest scientific instrument”. In: *UK Oracle User Group Conference and Exhibition (UKOUG), 14 - 17 November 2006* (Birmingham, UK), pages 414–416. URL: http://cds.cern.ch/record/1000757/files/ab-note-2006-046.pdf (cited on page 102).

[50] C. Roderick, R. Billen, and D.D. Teixeira. “Instrumentation of the CERN Accelerator Logging Service: Ensuring Performance, Scalability, Maintenance and Diagnostics”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), 10-14 October 2011* (Grenoble, France), pages 1232–1235. URL: http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/thchaust06.pdf (cited on page 103).

[51] C. Roderick, L. Burdzanowski, and G. Kruk. “The CERN Accelerator Logging Service - 10 Years in Operation: A Look at the Past, Present, and Future”. In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), 06-11 October 2013* (San Francisco, CA, USA), pages 612–614. URL: http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/tuppc028.pdf (cited on page 103).

[52] C. Roderick et al. "Accelerator Fault Tracking at CERN". In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17), 8-13 October 2017* (Barcelona, Spain), pages 397–400. URL: `http://jacow.org/icalepcs2017/papers/tupha013.pdf` (cited on page 153).

[53] C. Roderick et al. "The CERN Accelerator Measurement Database: On the Road to Federation". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), 10-14 October 2011* (Grenoble, France), pages 102–105. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/mopkn009.pdf` (cited on page 104).

[54] C. Roderick et al. "The LHC Logging Service: Handling Terabytes of On-line Data". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09), 12-16 October 2009* (Kobe, Japan), pages 414–416. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/wep005.pdf` (cited on page 103).

[55] P. Le Roux, R. Billen, and J. Mariethoz. "The LHC Functional Layout Database as Foundation of the Controls System". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'07), 15-19 October 2007* (Knoxville, Tennessee, USA), pages 526–528. URL: `http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/RPPA03.PDF` (cited on page 150).

[56] P. Le Roux et al. "LHC Reference Database: Towards a Mechanical, Optical and Electrical Layout Database". In: *Proc. of European Particle Accelerator Conference (EPAC '04), 05-09 July 2004* (Lucerne, Switzerland), pages 1882–1884. URL: `http://accelconf.web.cern.ch/accelconf/e04/PAPERS/WEPLT025.PDF` (cited on page 150).

[57] A. Rubini et al. "ZIO: The Ultimate Linux I/O Framework". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), 06-11 October 2013* (San Francisco, CA, USA), pages 77–80. URL: `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/momib09.pdf` (cited on page 68).

[58] A. Schwinn et al. "FESA3 – The New Front-End Software Framework at CERN and the FAIR Facility". In: *Proc. of Personal Computers and Particle Accelerator Controls (PCaPAC), 5-8 October 2010* (Saskatoon, Saskatchewan, Canada), pages 22–26. URL: `https://accelconf.web.cern.ch/accelconf/pcapac2010/papers/wecoaa03.pdf` (cited on page 75).

[59] J. Serrano et al. "The White Rabbit Project". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09), 12-16 October 2009* (Kobe, Japan), pages 93–95. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/tuc004.pdf` (cited on page 46).

[60] P. Sollander et al. "Alarms Configuration Management". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'07), 15-19 October 2007* (Knoxville, Tennessee, USA), pages 606–608. URL: `http://accelconf.web.cern.ch/AccelConf/ica07/PAPERS/RPPB03.PDF` (cited on page 111).

[61] A. Tovar et al. "Validation of the Data Consolidation in Layout Database for the LHC Tunnel Cryogenics Controls Package". In: *Proc. of International Conference*

on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'13), 06-11 October 2013 (San Francisco, CA, USA), pages 1197–1200. URL: `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2013/papers/thppc057.pdf` (cited on page 150).

[62] T. Wlostowski, J. Serrano, and F. Vaga. "Developing Distributed Hard Real-Time Software Systems Using FPGAs and Soft Cores". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), 17-23 October 2015* (Melbourne, Australia), pages 1073–1078. URL: `http://accelconf.web.cern.ch/AccelConf/ICALEPCS2015/papers/thha2i01.pdf` (cited on page 65).

[63] T. Wlostowski et al. "Trigger and RF Distribution Using White Rabbit". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), 17-23 October 2015* (Melbourne, Australia), pages 619–623. URL: `http://jacow.org/icalepcs2015/papers/wec3o01.pdf` (cited on page 47).

[64] Z. Zaharieva and R. Billen. "Rapid Development of Database Interfaces with Oracle APEX, Used for the Controls Systems at CERN". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09), 12-16 October 2009* (Kobe, Japan), pages 883–885. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/thp108.pdf` (cited on page 146).

[65] Z. Zaharieva, M. Martin Marquez, and M. Peryt. "Database Foundation for the Configuration Management of the CERN Accelerator Controls Systems". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'11), 10-14 October 2011* (Grenoble, France), pages 48–51. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2011/papers/momau004.pdf` (cited on page 145).

[66] M. Zerlauth et al. "The LHC Post Mortem Analysis Framework". In: *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09), 12-16 October 2009* (Kobe, Japan), pages 131–133. URL: `http://accelconf.web.cern.ch/AccelConf/icalepcs2009/papers/tup021.pdf` (cited on page 108).