DEGREE PROJECT

Evaluation of GUI technologies for CERN's Beam Interlock System

Tobias Skarhed

CERN-THESIS-2023-028
22/02/2023

Engineering Physics and Electrical Engineering, bachelor's level
2023

Luleå university of technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

[This page intentionally left blank]

Abstract

This thesis presents an evaluation of various graphical user interface (GUI) technologies for use in the supervision software of CERN's Beam Interlock System (BIS). The evaluation is necessary because the current GUI technology used by the supervision software, JavaFX, is no longer supported by Oracle. It also relies on an internal library that is no longer maintained. Furthermore, a second version of the BIS is being developed, and a GUI is needed that works with BIS and BIS2 in parallel.

Which technology is most suitable for the next version of the BIS supervision GUI?

This question was answered by gathering user stories from users and experts. Simple proof of concepts were developed for each technology, in order to identify technical limitations and register the development time. The user stories relevant to the choice of technology were listed, and each technology received points depending on whether they were able to fulfill the requirement or not.

Furthermore, in order to evaluate the ease of use, wireframes were created based on some of the user stories. These were qualitatively evaluated with stakeholders, which produced feedback that may be used for further development.

Out of the four technologies evaluated, PyQt scored 6, ACW scored 4, WRAP scored 6 and Grafana scored 3. WRAP and Grafana were discarded as viable options, since they were not able to fulfill hard requirements. ACW scored lower because it is web based and comes with a bigger maintenance overhead.

Keywords: BIS, BIS2, interlocks, UCAP, Grafana, PyQt, Angular, WRAP, fixed-display, control software, dashboard, GUI evaluation

Sammanfattning

Denna uppsats presenterar en utvärdering av olika tekniker för grafiska användargränssnitt (GUI) för övervakning av CERN:s Beam Interlock System (BIS). Utvärderingen är nödvändig eftersom den nuvarande GUI-tekniken som används, som är baserad på JavaFX, inte längre stöds av Oracle. Den beror även på ett internt bibliotek som inte längre underhålls. Dessutom utvecklas en andra version av BIS, därav behövs ett GUI som fungerar med BIS och BIS2 parallellt.

Vilken teknik är mest lämpad for nästa version av övervakningsprogrammet för BIS?

Denna fråga svarades by genom att samla user stories från användare och experter. Enkla proof-of-concepts utvecklades för varje teknik för att identifiera tekniska begränsningar och registrera utvecklingstiden. De user stories relevanta för valet av teknik listades, och varje teknik fick poäng beroende på om de uppfyllde kraven eller inte.

Skisser skapades baserat på några user stories, för att utvärdera användarvänligheten. Dessa utvärderades kvalitativt med intressenter, vilket genererade feedback som kan användas för vidare utveckling.

Utav de fyra teknikerna som utvärderades, fick PyQt 6 poäng, ACW 4 poäng, WRAP 6 poäng och Grafana 3 poäng. WRAP och Grafana gallrades bort då de inte kunde uppfylla de hårda kraven. ACW hade lägre poäng då det är webbaserat och kommer med mer underhållskostnader.

Acknowledgements

I could not have completed this thesis without these people, to whom I would like to extend my gratefulness.

Jonas Barth for teaching me the foundations of Java and initiating the BIS-UCAP project with me. Aleksander Buszydlik for his highly valuable contributions to BIS-UCAP during the summer of 2022. Jean-Christophe Garnier for giving good feedback and support during my thesis. The TE-MPE-CB team for reviewing my code and listening to my loud sighs in the office. The UCAP team, who provided invaluable support throughout the BIS-UCAP project. The rest of the BE-CSS for answering questions during the development. Ivan Romera Ramirez, Raffaello Secondo and Eric Veyrunes for giving feedback on mockups and proof of concepts.

Finally, I would like to thank Mercai de Ryan for making this period really fun.

Contents

Glossary

Accelerator software Commons Web (ACW) A web framework provided by BE-CCS. Packages are available for a Spring Boot backend and Angular frontend. 1, 13, 17, 18, 21, 24, 32

Architecture Level Modifiability Analysis (ALMA) A scenario based software architecture evaluation method with a focus on cost estimation. 4

Architecture Tradeoff Analysis Method (ATAM) Scenario based architecture evaluation method which has a closer connection to business drivers compared to its precursor, SAAM. 4

Controls Software and Services Group (BE-CSS) A group on the Beams (BE) department at CERN, responsible for providing software and services around the CERN control system. They provide frameworks such as JAPC, UCAP, ACW and WRAP. 1, 7, 9, 13, 17, 21, 32

Beam Interlock Controller (BIC) A controller board which is responsible for evaluating the status of user systems. 8, 10

Beam Interlock System (BIS) A system that propagates and evaluates user systems' status flags to make sure that the accelerator beam is permitted at the right place at the right time. A new version, BIS2, is under development. 1–3, 5, 33

Controls Configuration Data Base (CCDB) The database that stores the device information exposed to the CCDE. A REST API is available to query the database. 14

Controls Configuration Data Editor (CCDE) Web based device meta data editor. Possible to modify devices, device classes and NXCALS subscriptions. 6

Conseil Européen pour la Recherche Nucléaire (CERN) European Organization for Nuclear Research. 1

Controls Middleware (CMW) A family of technologies used for exposing and transporting device data, such as RDA3 and JAPC. The CMW devices service is used for looking up a device with the data stored in CCDB. 6, 13, 17, 31

General Purpose Network (GPN) The computer network that all CERN's office PCs are connected to. Access to this is required in order to reach certain services or websites. 22

Graphical User Interface (GUI) The visual software layer that users can interact with. 1

Java API for Parameter Control (JAPC) Java API for accessing device data. Built on top of RDA3. 7, 23, 25

Karlsruhe Architectural Maintainability Predictor (KAMP) A formal evaluation method for software architecture maintainability. Given a set of change requests, it is possible to generate a task list of the required actions needed to implement it. 4, 31

Large Hadron Collider (LHC) The last accelerator in the chain, running at the highest energies. Two particle beams run in opposite directions. Particle detectors are positioned at the collision points, where different experiments are conducted `https://home.cern/science/experiments`. 10

Next CERN's Accelerator Logging Service (NXCALS) Database service for storing device data. 9, 12, 25

Proton Synchrotron Booster (PSB) Ring-based accelerator that takes particles from LINAC4, accelerates them and extracts them to multiple other locations. 6

Proton Synchrotron (PS) An accelerator that may receive injections from either LEIR or PSB. Provides beam extractions into the East Area, SPS, the nTOF experiment and the Antiproton Decelerator. 6

Python Qt (PyQt) A Python wrapper around the C++ based Qt UI framework. 33

Role-Based Access Control (RBAC) An authentication and authorization service. A token is given to an authorized user which includes a set of roles. These roles can be checked against an access map when the user tries to perform an action on a device. 12, 23

Remote Device Access (RDA3) Protocol for device access, providing get, set and subscription operations. 6

Software Architecture Analysis Method (SAAM) A method for analyzing the viability of a software architecture based on scenarios. Created in 1993. 4

Super Proton Synchrotron (SPS) CERN's second biggest accelerator, responsible for injecting beams into the LHC in clockwise and counter-clockwise directions. The SPS also injects its beam into multiple other experiments. 6, 9, 10

Stateful Scalable Vector Graphics (SSVG) An extension to SVG that enables easy and portable animation between state changes. 13

Scalable Vector Graphics (SVG) A file format for rendering vector graphics. 10

Semantic Versioning (SemVer) A standard for how version numbers should be managed. Versions are on the form MAJOR.MINOR.PATCH. MAJOR defines breaking API changes. MINOR defines changes that are backwards compatible, such as extending functionality. PATCH defines backwards compatible bug fixes. An example of a version number could be 4.6.17. `https://semver.org/`. 32

Controls and Beam studies for Machine Protection team (TE-MPE-CB) Part of the Machine Protection and Electrical integrity group of the Technical department at CERN. Responsible for BIS control software, such as exposing device data to the CERN control system and its GUI. Lower level BIS responsibilities fall on the MI (Machine Interlocks) team of the same group. `https://mpe-cb.web.cern.ch/`. 1, 9, 32

Technical Network (TN) The computer network where production devices and computers for the control system are running. Access to this network is very limited, and internet access is not trivial. 22

Unified Controls Acquisition and Processing (UCAP) A framework for handling device data acquisition, transformation and publishing, through a set of Java or Python-based converters. 3, 7, 9

Web Rapid Application Prototyping (WRAP) Fixed-display dashboarding solution with access to CMW device subscriptions and NXCALS. 1, 3, 13, 17, 33

Database On Demand (DBOD) A service provided by the IT department that provides a UI for viewing and editing a given database. The IT department handles operations and backups of these databases. 12, 17, 32

Front-end computer (FEC) Responsible for communication with the hardware devices. 5, 8

accwidgets A PyQt component library provided by BE-CSS at CERN. 1, 3, 12, 23

User Permit An incoming status flag from the user system, which is used to evaluate beam interlocks. 8, 20

1 Introduction

Conseil Européen pour la Recherche Nucléaire (CERN) has used particle accelerators to study subatomic particles for the last 60 years, with one of the more recent contributions being experimental proof that the Higgs boson exists. Other particles interact with the Higgs field, which gives them mass. This is the first experimental proof where mass comes from. The experiments have been essential for developing and proving the standard model in physics.

A series of particle accelerators are required to enable the collision of the high energy particles. This is done in a stepwise manner, where the next accelerator in the chain increases the speed at which the particles travel. Along the accelerators, there are multiple experiments placed, with different research goals in mind.

CERN's Beam Interlock System (BIS) is vital for the safe operation of the accelerators. It serves as a line of defense against fault in the accelerators' systems. BIS gathers a set of flags from user systems, that describe their status. It makes a decision based on the information from the user systems whether the beam is permitted in a specific part of the accelerator. A failure to react quickly to a user system permit change may prohibit extraction of a high energy accelerator beam, which may cause severe damage to equipment, long downtime and costly repairs.

The current BIS supervision software is based on JavaFX and has deprecated dependencies, which are no longer supported at CERN. This solution will be replaced by a data acquisition pipeline and a new Graphical User Interface (GUI). They are intended to serve the next version of BIS: BIS2. [1]

Controls Software and Services Group (BE-CSS) develops and maintains software frameworks for the accelerators' control system. Two of these are Accelerator software Commons Web (ACW) and the library for PyQt called accwidgets. They also provide a dashboarding solution called Web Rapid Application Prototyping (WRAP).

Grafana is also evaluated, as it is a widely used technology at CERN, although most often in the context of IT infrastructure monitoring.

The question that will be answered in this thesis is which GUI technology is most suitable for the next BIS supervision GUI, and which one fits best with the Controls and Beam studies for Machine Protection team (TE-MPE-CB).

These technologies are evaluated on these conditions:

- Ability to fulfill technical requirements
- Maintainability for the responsible software engineers
- Ability to implement views that are required by users
- Ease of use

User stories are used in order to gather information about requirements from the stakeholders. A set of stories are identified as affecting the choice of technology, and the technologies score points depending on their ability to fulfill each user story.

Wireframes are drawn based on some user stories, which improves upon shortcomings of the previous BIS GUI. Users provide feedback on these, as well as some proof of concepts that have been created for each technology.

This thesis will provide a base for further development of the next BIS supervision GUI, provides a deeper insight into the needs of stakeholders and a throughout comparison of different GUI options available at CERN.

1.1 Delimitations

This thesis will not investigate the full user experience for the GUIs, because doing four parallel implementations is not feasible and the time is limited. The ease of use case will be restricted to wireframes for a few use cases.

Only high level software architectures will be created, in order to focus on the amount of services and configurations to be maintained.

There will be no in-depth analysis of graphing time series data with nanosecond timestamps, as it is not directly related to the choice of technology. An evaluation of this is more appropriate once a technology has been selected.

1.2 Contribution to the field

The proposed evaluation compares different UI technologies, something which has already been done in multiple contexts. It has not been done in the context of BIS. Another addition is the comparison of technologies with the help of a scoring system based on stakeholder input. Previous research has mostly discussed pros and cons, and drawn upon previous experience. In general, none of the previous evaluations take users into account, but rather focus on longevity and maintainability of the applications.

This thesis serves as an important part in the development of the supervision software for BIS2

1.3 Thesis overview

Section 2 describes available research on GUI evaluations, BIS and some alternative methods for maintainability evaluation. Section 3 covers the CERN control system, describes how BIS works, lists available GUIs, stakeholders, IT services provided by CERN and finally describes the different GUI technologies that are going to be evaluated. Section 4 describes the tools used for the evaluation; user stories, architecture diagrams, proof of concepts and mockups. It also lists the user stories that is the basis for comparison of technologies. Section 5 shows how the different technologies were able to fulfill user stories or not. It also presents wireframes and feedback from operators and experts. Section 6 describes implemented proof of concepts for each technology in Section 3.6 and their suggested architectures. Some implementation issues are also briefly discussed. Section 7 concludes the thesis by answering which technology performed best. It also compares the method to more formal ones, evaluates the value of mockups, and discusses technical issues in depth. Areas for future work are also discussed.

2 Related work

This section covers work related to GUI evaluations at CERN, the current developments related to BIS and different methods for evaluating maintainability.

2.1 At CERN

BIS2, the next version of BIS is described in [1]. It mainly covers the hardware upgrades and describes how BIS2 works. The proposed evaluation of the GUI technologies for the BIS supervision extends to BIS2 as well as BIS, as the upcoming GUI and data pipelines will serve both systems.

[2] describes the previous data acquisition and GUI for BIS. This approach assesses the relevance of reactive streams in the context of accelerator controls. One of the next steps described would be to use streams as a service to remove the data processing from the client layer. This approach is no longer viable, as a core library that bridges CERN's JAPC protocol to reactive streams, is no longer maintained.

A pipeline using the Unified Controls Acquisition and Processing (UCAP) framework has been developed for the purpose of removing data processing from the client and replacing the old Java streams-based solution. Data from these pipelines will be consumed by the GUIs mentioned in this thesis, which builds upon the further work mentioned in [3].

An evaluation of PyQt and ACW as replacements for JavaFX are presented in [4]. One of the conclusions is if a web technology is to be adapted long-term, one has to expect to rewrite the application during its lifetime, as web frameworks are prone to change. The conclusion was to use PyQt and rapid application framework (RAD) for control applications and web applications, mainly for fixed displays and dashboards. Since then, RAD has had a decline in maintenance and pre-made dashboard solutions have become more prevalent. This thesis does a similar evaluation, but in the context of the Beam Interlock System. It also covers more recently developed and updated technologies, such as a newer version of ACW, CERN's accwidgets library for PyQt and CERN's new dashboard application for the control system, WRAP. Furthermore, a third party dashboarding solution, Grafana, is also considered. The comparison in this thesis uses a scoring system to compare the different technologies.

An example of how a Python Qt UI may be implemented in the context of CERN's control system is found in [5]. It presents the development workflow and lessons learned from implementing a new timing sequencer application. The implementation uses a 3-tier architecture, where the PyQt application is used for the GUI, which communicates with RESTful Java services and the Central Timing process. It further discusses setup for continuous integration and deployment. Some issues they experienced with Python and PyQt were dynamic typing and lack of compilation checks. This thesis will only develop a proof of concept of a GUI for the BIS use case, and will not cover a three tier architecture or consider continuous integration and deployment to a great extent. Once a technology has been chosen for the BIS supervision GUI, some of these things may be looked into. This thesis will not go as in depth implementation-wise and focuses more on comparing PyQt with other technologies, weighing their viability.

In [6], there is an example for how Grafana is already used in NetIs for monitoring data acquisition for CERN's ATLAS experiment. The use case is similar to BIS to some extent, as it concerns visualizing time series data. They had further requirements for a tree visualization, which was not supported by any available panels. In this case they developed a custom panel, but rather than using Grafana's frontend API, they injected a script element via a text panel. This improved the ease of maintenance, since only knowledge of web and JavaScript are required. In this thesis, only supported approaches are used, with the caveat of using third party plugins. Time series visualizations will not be implemented in the proof of concepts either.

2.2 Methods for maintainability evaluation

Having maintainable software is important both from a developer experience and business point of view.

There is a set of methods available to evaluate the maintainability of systems:

- Static code analysis
- Architectural modifiability analysis
- Test coverage

As this thesis evaluates technologies and programs that have not been implemented yet, static code analysis and test coverage are not applicable. Doing an analysis on an architectural level is a viable option, and may be a good candidate for comparing technologies and the different architectures they require. The main problem

with analyzing the architecture is that the user is not necessarily in focus in the same way as other agile methods.

2.2.1 Scenario based architecture evaluation

Given a system architecture, how does it fulfill a usage scenario? An initial method for identifying bottlenecks in system architecture was created in the 1990s, called Software Architecture Analysis Method (SAAM). It has later been superseded by Architecture Tradeoff Analysis Method (ATAM). [7]

Both of these methods provide a way to analyze the modifiability of software architectures based on user scenarios. The user scenarios are also prioritized. Given a change scenario, an analysis is made to figure out how much needs to be changed. In the case of ATAM, the process is done with a progressively wider audience, and the changes are documented categorically. A side effect of these methods is good documentation of why certain architectural decisions have been made.

A method that takes the approach a bit further is Architecture Level Modifiability Analysis (ALMA). Here, scenarios are given weights and cost estimation for changes are made. This is good for longer term resource planning.

The scenario-based approaches heavily rely on expert estimations and experience, which may give highly varying results that depend on domain experience.

2.2.2 Formal architecture analysis

Karlsruhe Architectural Maintainability Predictor (KAMP) is a metamodel which can be applied to different domains. In this scenario, KAMP4IS (KAMP for Information Systems) is most applicable. [8]

For this method, an architecture model is created, with the possibility to add context information, such as source files, build configurations and deployments. These are used to create a context annotation model, which enhances the architecture model. Change requests are provided to the architecture, and a list of tasks are generated, which are required in order to fully perform the change.

With this approach, it is easy to quickly compare the modifiability of multiple architectures, as a comparison of the task list lengths can be made.

This method is more scalable than the scenario-based ones, as it is possible to significantly increase the size of the architecture. It also extends modifiability to not only cover source code, but also the surrounding services that are required for maintenance.

However, due to lack of documentation and a bad user experience, it is proven to be very difficult to use, and does not make sense outside of academia.

3 Background

This section covers CERN's control system, what BIS is and how it integrates into the control system. It also presents the GUI technologies that will be evaluated and CERN services that may be used by them.

3.1 CERN's control system

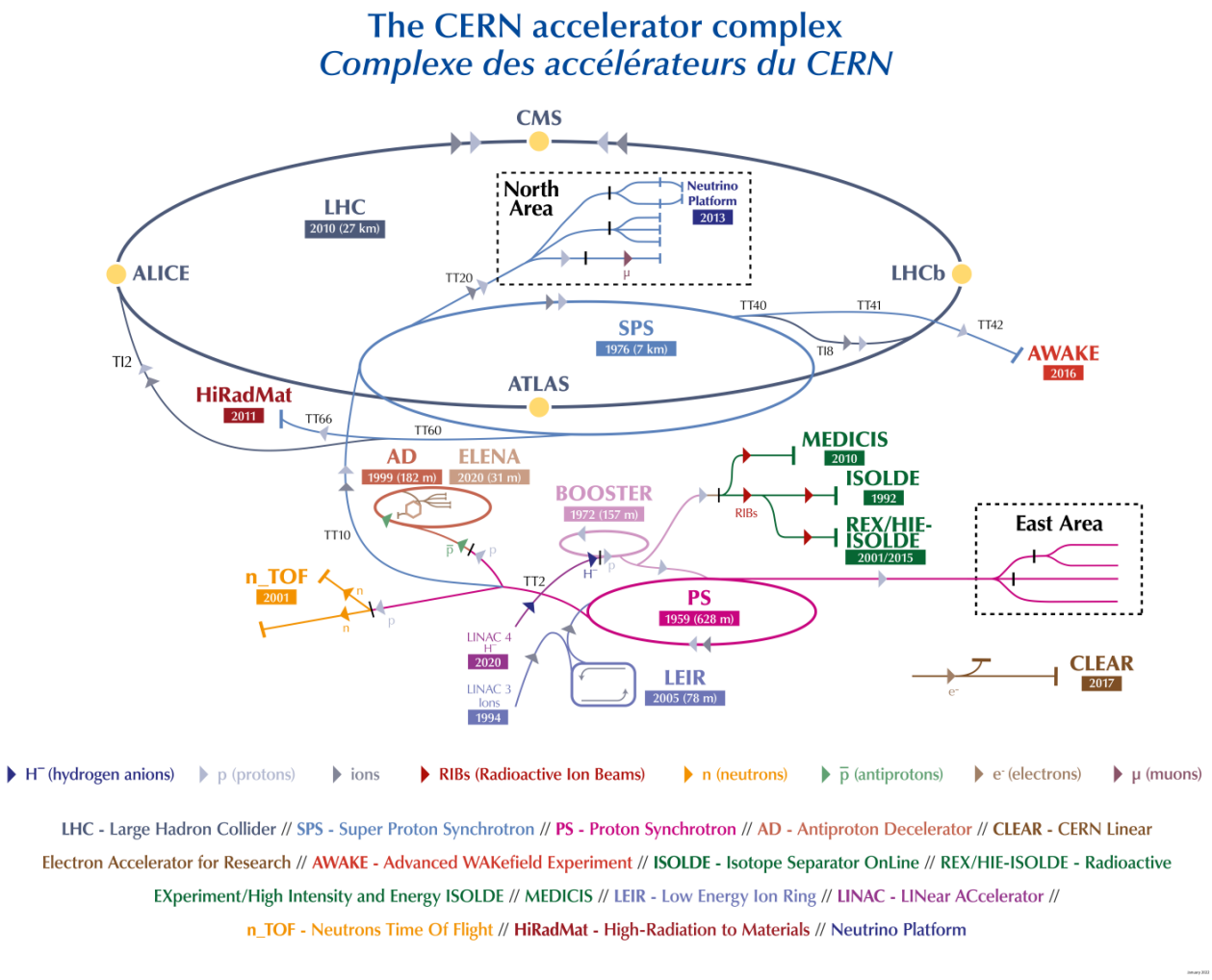


Figure 1: CERN's accelerator complex consists of multiple accelerators, increasing beam energy stepwise. Along the accelerator lines, there are multiple experiments that the beam may be extracted to. ©CERN

CERN has eight accelerators and two decelerators, as seen in Figure 1. In order to monitor and operate these, there is a need for a scalable control system. CERN's control system software operates on three different levels: [9]

1. **Front-end computer (FEC)**, responsible for communicating with a set of hardware devices.
2. **Back-end server**, computers running high-level services, responsible for advance business logic aggregating information from multiple systems, and providing data to GUIs.
3. **Control-Room computer**, or consoles. This is where operators can see and interact with device and system information.

This thesis mainly covers software that runs on control room computers, office PCs and back-end servers.

3.1.1 Device-Property model

All device names in the control system follow the same model. This is relevant in order to understand what the devices are seen in different screenshots. The device-property model have three levels to it:

1. **Device name**, which is an entry point to communicate with a virtual or hardware device.
2. **Property name**, a logical grouping of values. These could be acquisition properties, setting properties or commands.
3. **Field name**, which maps to the respective value. These values only support primitive types.

The naming convention for devices is based on the system it is related to, its location and what purpose it serves. As an example, the device name *CIBM.400.LN4.L4T* refers to a CIBM board that is located in building 400, for the accelerator LINAC4 in the transfer zone (L4T). *CIB.USC55.R5.B2* refers to the CIBM board located in the *USC55* location if the LHC, in the *R5* position for the second beam (*B2*).

3.1.2 Accelerator cycles and selectors

Some accelerators can extract their beam to multiple targets, such as another accelerator or an experiment. In order to differentiate values from each other, the cycle bound accelerators – such as Super Proton Synchrotron (SPS), Proton Synchrotron Booster (PSB) and Proton Synchrotron (PS) – defines a *cycle* as a multiple of 1.2 seconds. Each cycle may relate to a user, which is the name of the target accelerator beam.

This information is important, since it is possible to understand how accelerator devices operate with different targets, and to make sure the right cycle extracts beams to the correct destination.

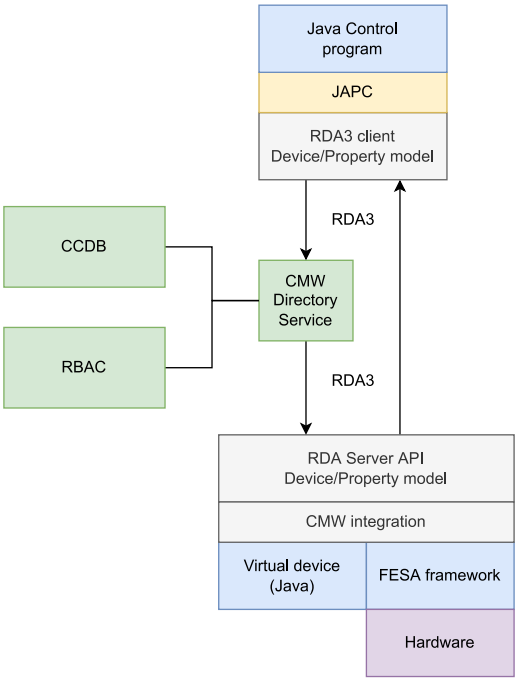


Figure 2: The Commons Middleware stack with Java control software at the top. This uses JAPC to interact with the RDA3 client code. The request for the device is then looked up via the CMW directory service, and authorized with RBAC, before being forwarded to the RDA3 server that serves the data. The RDA3 server may serve data provided by a FESA class or a virtual device implemented in Java. This is not the full extent of the CMW stack, as it only contains the components relevant to UCAP and Java-based control software.

3.1.3 RDA3

Remote Device Access (RDA3) is the third version of a protocol built on top of TCP/IP with the ZeroMQ messaging library. [10] It enables synchronous and asynchronous get and set methods, while also allowing subscriptions to devices. There is a command line tool for these operations, which makes troubleshooting and verification easy.

RDA3 handles name resolution and communicates with the centralized Controls Middleware (CMW) directory service, which means that specifying which server the device is hosted on is not necessary. For a device to be centrally accessible, it needs to be registered with Controls Configuration Data Editor (CCDE). RDA3 is the glue that keeps everything together in the CMW stack, as seen in Figure 2.

3.1.4 JAPC

Java API for Parameter Control (JAPC) is a Java library on top of RDA3. It can be used for other protocols such as JMS too, but that is out of scope for this thesis. A JAPC parameter is a combination of a device name and a device property, according to the Device-property model. It is also possible to specify which field to be operated on. Under the hood, the get request that is sent to RDA3 always returns all fields for a property. [11]

An example of a JAPC parameter is *CIBM.400.LN4.L4T/BoardRegisters#registerNames*. This refers to the *CIBM.400.LN4.L4T* device and the *registerNames* field of the *BoardRegisters* property.

3.2 UCAP

UCAP is a framework developed by BE-CSS for processing device data, based on JAPC. [12] It serves the common use case of data acquisition, transformation and publishing, similar to an extract-transform-load (ETL) pipeline. It uses a polylithic architecture. This means that the running code is decoupled into tiny modules.

These small modules are called converters. A well-written converter is very generic, and its functionality is decided and modified with a configuration. A converter with a given configuration is called a transformation.

The transformations are run on UCAP nodes, which infrastructure is maintained by BE-CSS, hence decreasing the maintenance overhead for teams maintaining UCAP pipelines. The UCAP node architecture can be seen in Figure 3

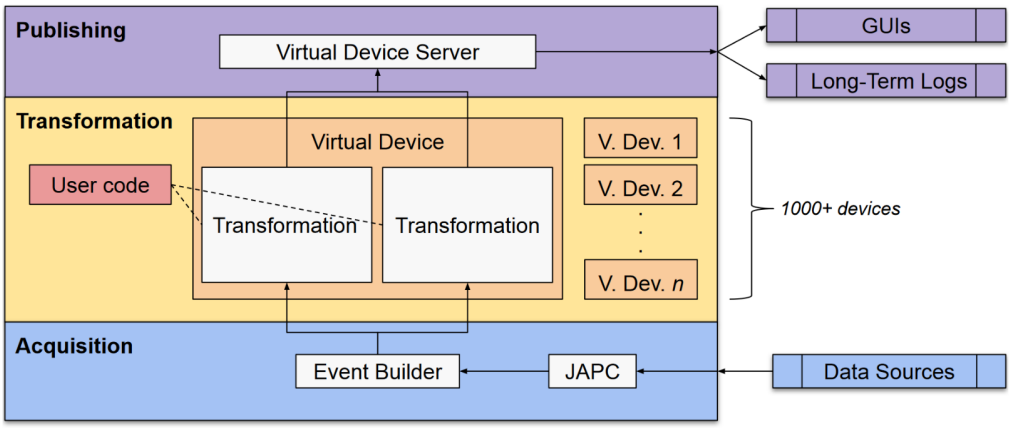


Figure 3: UCAP node architecture. Data acquisition is in blue, passing data onto the virtual devices that run transformations (yellow). The output is then republished and made available to other sources (purple). ©CERN, Cseppentő and Büttner

3.3 Beam Interlock System

The purpose of the Beam Interlock System is to provide a single interface to report status of different user systems, which then can be used to determine safe beam injections and beam dumps. The first version of the Beam Interlock System has had 15 years of excellent operation. An upcoming upgrade to BIS2 is planned., Because of the increased demand for more systems to be connected and hardware reaching end of life. [1]

The Beam Interlock System consists of a set of interconnected Beam Interlock Controller (BIC) boards, as seen in Figure 4. These configurations are deployed in accelerators and transfer lines. The BICs works together to evaluate whether the beam is allowed in specific parts of an accelerator or a transfer line. Each configuration propagates a global beam permit to either a beam dumping system or a beam transfer system, depending on whether it is a ring configuration or a tree configuration.

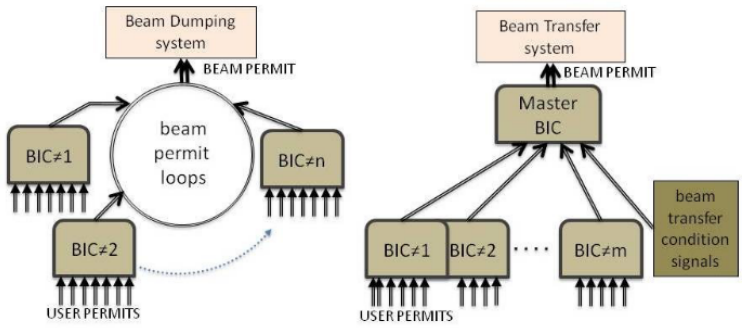


Figure 4: Two different BIC configurations of the BIS. Beam permit loops for ring configuration to the left and a tree configuration with a master BIC to the right, which is used in transfer lines. ©CERN, Puccio et al.

3.3.1 The device manager

A BIC takes 14 User Permits from user systems to evaluate a local permit. These User Permits have two channels, A and B, for redundancy. The local permit flag is then used throughout the rest of the interconnected BICs in order to evaluate the global beam permit.

There are two types of BICs. *CIBM* is the most common one with a less complex use case, taking user permits from user systems. *CIBX* is a similar board that can be used as a master BIC in a tree configuration, as seen in Figure 4. *CIBX* uses inputs from *CIBMs* in order to evaluate the global beam permit.

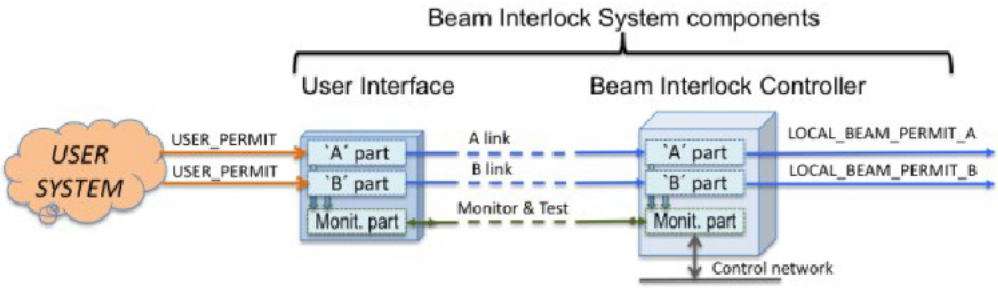


Figure 5: Overview of the BIS components. User Interface refers to the *CIBU* board, which take two cables for a single user permit. The BIC provides a connection to the control network, which is used for monitoring. ©CERN, Puccio et al.

Through the monitoring part of the BIC, as seen in Figure 5, device data is served to a FEC. The data is then republished through RDA3, which makes it accessible to consume via applications on the back end servers.

Some of the data exposed from the BIC:

- Local, global and software permits
- User permits and masks with microsecond precision
- Disabled input states

- Device status flags
- Input glitches
- Frequencies

3.3.2 BIS-UCAP

The coming from the devices do not have a format that is easy to consume and quickly understand. In order to display it in UIs and write to a logging database, the data first needs to be processed. Data processing pipeline have been constructed with UCAP for this purpose. [3]

The BIS-UCAP pipelines serves as a replacement for the old service that served as a proxy between JAPC and Java streams, which now has deprecated core dependencies and can no longer be maintained.

Some benefits of using UCAP for the pipelines are:

- Infrastructure maintenance offloaded to BE-CSS
- Clear separation of concerns between data transformations and UIs
- UCAP adoption has increased in recent years and seems to be a viable long term bet

The data published from UCAP is displayed in GUIs and is written to Next CERN's Accelerator Logging Service (NXCALS).

Furthermore, operations have developed a set of UCAP converters that expose active interlocks for an accelerator, which is used by the BIS monitor GUI.

3.3.3 BIS Streams GUI

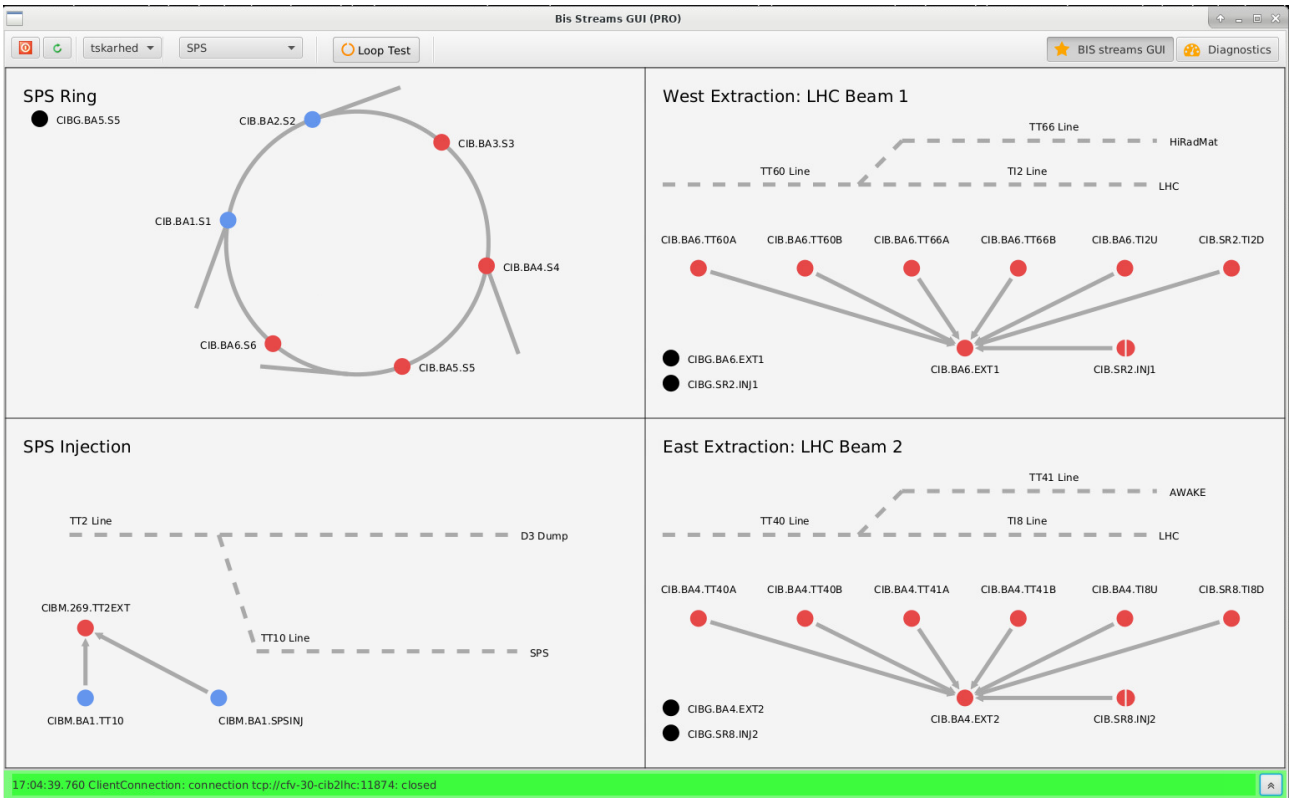


Figure 6: A view in the BIS Streams GUI that displays the status of the BICs in the SPS, with different visualizations for the transfer lines and the accelerator ring.

BIS-streams is the current GUI responsible for displaying data related to the Beam Interlock System. It is developed and maintained by TE-MPE-CB. It provides simple visualizations of the different accelerators, where BIS devices are located and their current status, as seen in Figure 6. It is also possible to go more in depth by clicking on the dots representing the devices.

The user interface is written in JavaFX, Scalable Vector Graphics (SVG) images are used for accelerator visualizations and RxJava for data transformations. [2]

3.3.4 BIS monitor

BIS monitor provides a set of GUIs specifically for Large Hadron Collider (LHC) and SPS. It is developed and maintained by operations. The LHC use case provides monitoring of BIC devices in the LHC ring, and the injection points for beam 1 and 2 with permit and masks information. It is possible to set masks for devices en masse based on some parameters, something which is not available in BIS-streams.

These are the BIS monitor UIs available for the LHC:

- BIS Monitor, responsible for the device in the LHC ring
- BIS TI2 Monitor, responsible for monitoring beam 1 injection from the SPS
- BIS TI8, responsible for monitoring beam 2 injection from the SPS

As the SPS has more extraction points, it also has more BIS Monitor UIs to manage these:

- SPS Ring BIS Monitor
- TT10 BIS Monitor
- LHC Beam 1 BIS Extraction Monitor (Same as TI2 for the LHC)
- LHC Beam 2 BIS Extraction Monitor (Same as TI8 for the LHC)
- HIRADMAT Extraction Monitor
- AWAKE Extraction Monitor

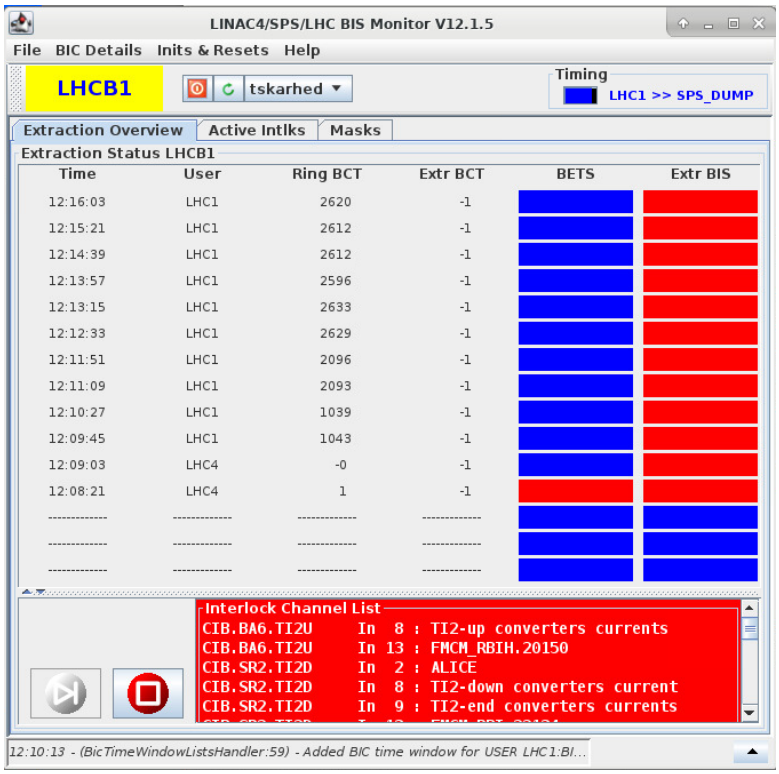


Figure 7: Extraction status of beam 1 from SPS into LHC. A list of the interlocked channel is provided at the bottom. An equal UI is provided for beam 2 injection. In this case, information about other systems is also displayed, such as the BCT.

The BIS Monitor GUIs fall into two categories: extraction status and BIC permit status centric. The GUIs responsible for extraction status takes additional data into consideration, which is not provided by the BIS (Figure 7). The GUIs displaying BIC status has a main overview that display all updates and their corresponding user, as seen in Figure 8.

BIS-streams provide multiple views that contain the same information as all the different BIS monitor GUIs.

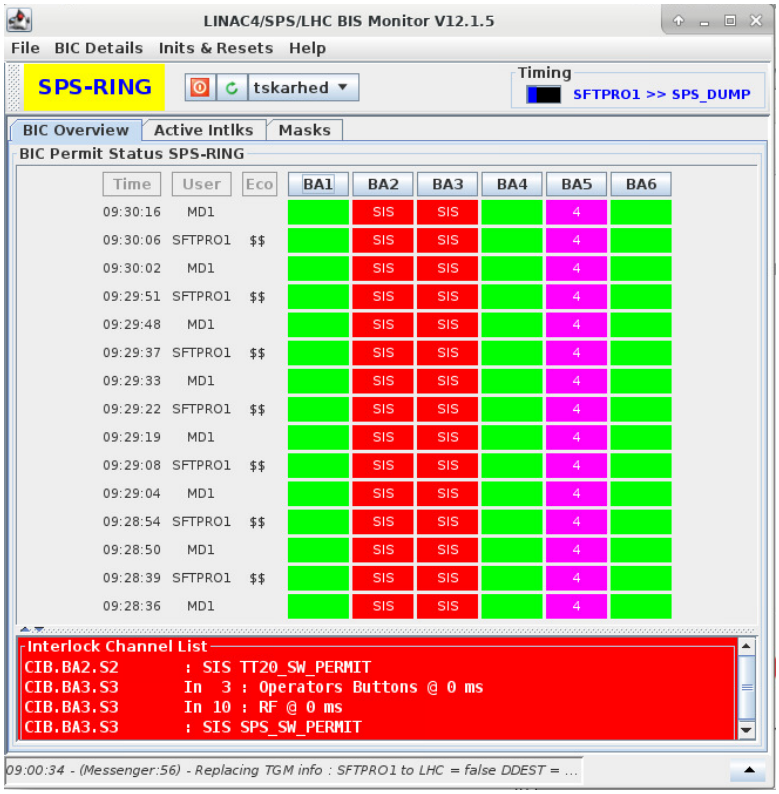


Figure 8: BIS monitor for the SPS ring. The overview shows the devices and their current interlock status, along with the user update. Red is used to display which device is interlocked by which system. Purple is used for the caveat where the SBDS channel is supposed to be false. If the input becomes true, it turns red.

3.4 Stakeholders

There are multiple stakeholders for the BIS GUI:

- LHC/SPS operators
- LINAC4/PSB operators
- Hardware experts
- Software developers
- Product owner

The operators for different accelerators have different needs. These may overlap to some extent, such as requiring available data for specific cycles and having a specific device overview. The software developers and their supervisor are also considered as stakeholders, as maintaining the software may cause an increased workload, which is undesirable in a team that is stretched thin.

3.5 Available CERN services

The multiple departments at CERN offer a wide variety of services. They provide services and operates the required infrastructure in order to take the software operations workload from the team developing a certain application.

3.5.1 Database On Demand

Database On Demand (DBOD) is a database hosting service. It offers MySQL, PostgreSQL and InfluxDB instances. Maintenance, such as backups, recovery and updates, is done by the IT department. The end user is provided with a user interface to create, configure and manage their database instance.

3.5.2 RBAC

Role-Based Access Control (RBAC) is a system for user authorization and authentication. It is not a security measurement for malicious users, but rather there to prevent a user to accidentally making the wrong action, which may cause accelerator downtime. It uses tokens for session management and access maps in order to authorize device actions. [14]

3.5.3 NXCALS

NXCALS is a database responsible for logging device data. It integrates well with the rest of the control system, with the device-property model for writing fields into the database. It is based on Hadoop and HBase. [15]

3.5.4 Web hosting

At CERN, there are multiple options for hosting web applications. If the team in question has access to their own hardware, that's an option.

There are two services provided by the IT department. They host an OpenStack instance, which makes it very easy to create virtual machines with a given base image. The IT department also provides an OpenShift instance.[16] It is a platform for managing containerized applications, which is good for maintaining high availability.

A self-service Grafana operator is available on CERN's OpenShift instance.[17] This makes it easy to integrate Grafana with CERN's SSO and map Grafana roles (Viewer, Editor and Admin) to specific CERN E-groups.

3.6 GUI Alternatives

The CERN control system implements two kinds of GUIs:

- **Controls applications** that are made for interactions, such as querying for data and interacting with devices. These are used by operators and experts.
- **Fixed displays** made for displaying data on a dashboard in a control room or online.

This thesis will explore four different alternative frameworks for a new GUI; two frameworks for fully fledged controls applications (PyQt and ACW) and two alternatives that traditionally are use for fixed displays (WRAP and Grafana).

3.6.1 PyQt

PyQt is a Python user interface framework, wrapped around the C++ based Qt. It provides an API with basic UI widgets that can be orchestrated into larger applications. Widgets provided by Qt follows a Model-View architecture, where the data may be separated from the view logic.

At CERN a library of components, called accwidgets, has been developed in order to handle regular use cases, such as RBAC integration, displaying application logs and graphing.

There has been previous implementations for using PyQt as an GUI alternative at CERN [5], and this experience may be useful to rely upon.

Software deployed with acc-py ends up on NFS.

3.6.2 Accsoft Commons Web

ACW is a framework for developing web based GUI applications. It has a frontend written in Typescript with the Angular framework. Angular templates are used for component structure and layout, while a component class is used to populate it with data and add functionality. Angular relies heavily on dependency injection, where services may be injected into components to access external functionality and increase separation of concerns. Data streams and requests are managed with RxJS.

The backend is written with Spring Boot, and provides multiple packages for authentication and authorization. A database is required in order to enable these features. Spring Boot may be used with the frontend without ACW functionality. ACW is maintained by BE-CSS. [18]

3.6.3 Web Rapid Application Prototyping

Figure 9: A dashboard in WRAP with some data from LHC and experiment operations.

WRAP is a fixed-display solution that is tightly integrated with the CERN controls system. Some desirable features:

- Live data from CMW devices, such as UCAP
- Tight integration with CCDB for searching for devices and properties
- Dynamic dashboards based on URL parameters
- Integrated CMW set functionality

When creating dashboards with WRAP, you are provided with direct access to variables from NXCALS. [19]

With WRAP being fit to the CERN control system use cases, the availability of widgets are a bit different compared to Grafana. A format for handling SVG animations, called Stateful Scalable Vector Graphics (SSVG) have been developed [20], which may prove useful for complex visualizations.

3.6.4 Grafana

Grafana is an open source dashboard solution, with a plugin-based data source system. This allows the users to visualize data from multiple different sources. Some departments at CERN already have Grafana instances deployed for monitoring. [15].

Some desirable features with Grafana:

- Dashboard provisioning enables GitOps, which simplifies deployments
- Easy to add new views

- Available panel plugins for tables, graphs and SVG images
- Integration with external single-sign on.

Grafana offers an extensive plugin system [21], with three kinds of plugins:

- **Data source plugin**, responsible for reading data from a given database (e.g. MySQL, Prometheus, InfluxDB)
- **Panel plugin**, responsible for displaying data. This could be in the form of an image, map, graph, or a pie chart.
- **App plugins** bundle other plugins and provides dashboards that make a cohesive experience. These plugins end up in Grafana's sidebar.

Implementing a custom plugin is relatively easy, since Grafana provides both extensive documentation and example plugins. [22]

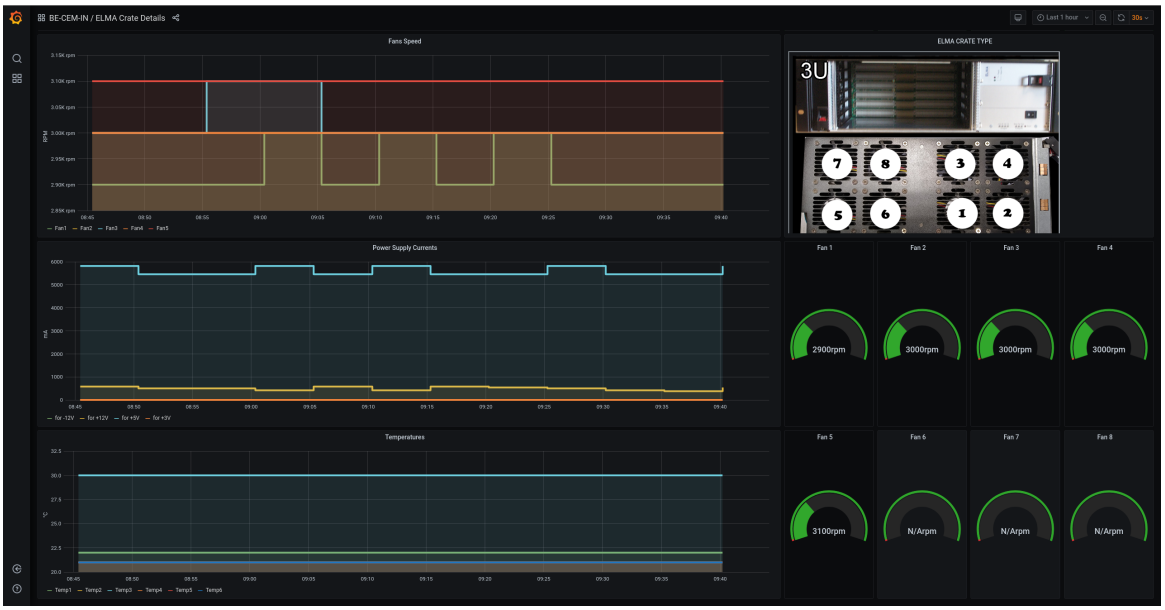


Figure 10: A dashboard in COSMOS' Grafana instance with power supply metrics.

Grafana is used extensively at CERN for server and application monitoring with the COSMOS system. [23] An example of a dashboard can be seen in Figure 10.

3.6.5 Dashboard provisioning

The available dashboard will depend on the devices available in Controls Configuration Data Base (CCDB). Therefore, a solution for generating dashboards based on API calls to CCDB is required. Grafana offers a solution for pre-configured dashboards, where they are stored as JSON files on the file system. These JSON files are continuously checked for updates, and loaded into Grafana if they change.

This is good for two reasons: deployment of updated dashboards can happen with no downtime, and dashboards can be dynamically generated. One approach for dashboard generation is to use Jsonnet [24], a templating language for JSON.

Instead of using dashboard provisioning, it is possible to use a custom REST data source that can make requests to CCDB on a dashboard basis. This would require a third party plugin, and may not end up being very reliable compared to dashboard provisioning.

4 Method

This section presents the methods and tools used for evaluating and comparing the GUI technologies.

4.1 User stories

User stories are a common to gather user requirements in agile software projects. Their purpose is to highlight the added value a user might be looking for, rather than a specific feature they are requesting. A user story is written on the form «As a [stakeholder], I want [feature], so that [added value]».

These are designed as a way to promote conversation between stakeholders and developers. An example where it could be useful is if a user asks for a button. The important thing is not the button itself, but rather what it does. Through a conversation with the developer, an alternative solution might be more suitable. It could be for technical or usability reasons. Maybe a button could not directly be implemented in a user-friendly way, so a context menu with actions through a right-click might serve the user better.

4.1.1 Gathering user stories

User stories were gathered by talking to the stakeholders listed in Section 3.4. Some user stories were also extrapolated from existing GUIs and verified with stakeholders. Developer user stories were created based on the team's previous experience, and the experience gained during development of the proof of concepts.

4.1.2 Identifying themes and hard requirements

The user stories were sorted into groups of related stories, each called an individual theme. The themes can be seen in Appendix A. This was done in order to get a better understanding of the areas which need development, and may serve useful for further development. In terms of the evaluation, the themes have no impact.

A set of user stories that had a direct impact on choice of technology were identified. A way to evaluate each story was selected. From these stories, a subset was selected as hard requirements. These were stories that were either possible to support or not. If these stories could not be supported, the technology was discarded.

The other identified stories were either binary in their support, or provided a range, where either a bigger or smaller value was better.

4.1.3 Scoring

Each technology received a point for each story they could either support (yes or no) or if they scored best on the range for that particular story.

The points from the stories were summed up, and the hard requirements decided if the technology was a viable option or not, regardless of the score.

The development time for the proof of concepts was also taking into account with the scoring.

4.2 Process for generating developer user stories

This section covers architecture diagrams and proof of concepts, which were tools that were used to generate developer user stories and evaluate them.

4.2.1 Architecture diagrams

In order to get a better understanding of the amount of services and data flow, architecture diagrams were created. These serve as examples for how a production-like application might look. It is also a good way to get an overview of the complexity of a system.

4.2.2 Proof of concepts

Proof of concepts were developed in order to evaluate the time required for initial development, and to identify other pain points. During this process, a lot of developer user stories were developed. The purpose of the proof of concepts is not to have a good end product, but rather to get some familiarity with the technologies.

The proof of concepts provided a single use case, and did not implement the entire archirtecture shown in the architecture diagrams.

4.3 Mockups

In order to improve the experience compared to BIS-streams GUI, mockups were created to address some of the user stories.

Mockups provide a low-cost investment for initial GUI development, where layouts can quickly be improved iteratively.

The mockups were evaluated qualitatively by showing them to stakeholders and asking for feedback.

5 Result and evaluation

This section covers the technology comparison based on user stories and wireframes, with evaluations based on ease of use.

User stories were gathered by meeting stakeholders and identifying use cases from available GUIs. Stories from the developer point of view were discovered by developing proof of concepts. All user stories can be found in Appendix A.

5.1 Comparison

The user stories relevant for comparison were chosen according to Section 4.2. Other stories were deemed irrelevant, either because they affected all technologies the same or that they did not depend on the choice of technology at all.

The different user stories were evaluated in the following manner:

- **Development time**: roughly estimated while developing the proof of concepts in Section 6.
- **Maintained services and clients**: amount of internally managed services were counted in the architecture diagrams in Section 6. (User Story A.2.7)
- **Manageable development environment**: if it was relatively easy to set up the development environment during the proof of concept development. This is also reflected in the development time. (User Story A.2.5)
- **Dependencies**: counted from dependency files in the projects. For Grafana, external plugins were also considered dependencies. (User Story A.2.4)
- **Support from BE-CSS**: whether BE-CSS provides official support for the technology.
- **Micro/nanosecond timestamp support**: research in technology issues. WRAP already integrates with CMW, which means the timestamp format is supported. (User Story A.4.1)
- **Dynamic and flexible**: conclusions from the proof of concepts whether the technology can be easily scaled to multiple devices and accelerators. (User Story A.5.4)
- **Complex set actions**: if the technology has the capability to either implement complex set actions or make calls to a service that can. This was investigated during proof of concepts development. (User Story A.1.6, User Story A.1.7, User Story A.1.8)

In Table 1 the technologies are compared based on the points identified in Section 4.2. A technology scores one point for each user story it manages to fulfill. Hard requirements are able to disqualify a technology, as these serve as the bare minimum.

Requirements	ACW	PyQt	WRAP	Grafana
Development time	1 month	1 week	**4 hours**	4 hours
Maintained services and clients	2 (3)	**1**	**1**	2 (3)
Manageable development environment	No	**Yes**	**Yes**	**Yes**
Dependencies	50+12	6	**0**	3+12
Support from BE-CSS	**Yes**	**Yes**	**Yes**	No
Micro/nanosecond timestamp	**Yes, with library**	**Yes, with library**	**Yes**	No
Dynamic and flexible	**Yes**	**Yes**	No	**Yes**
Complex set actions	**Yes**	**Yes**	No	**Yes**
Score	4	**6**	~~6~~	~~3~~

Table 1: Comparison between technologies based on how they fulfill different requirements. Hard requirements are in cursive. Value that are bold signify that the technology either managed to fulfill the requirement or that it had the best score. Technologies with a strike-through score did not manage to fulfill all hard requirments.

Dependencies for web-based solutions were split into frontend and backend. The amount of services for ACW and Grafana depend on whether DBOD is counted or not. The Spring Boot backend for ACW was reused for the Grafana implementation, therefore the 4-hour development time comes with a caveat, hence Grafana did not score for that requirement. WRAP and Grafana were discarded since they do not manage to fulfill all hard requirements.

The development time for ACW was long because of dependency issues with the development environment. Once they were resolved, the backend took 3 days and the frontend 1 day.

Section 7.3 has a more in-depth discussion about the different technologies.

5.2 Wireframes and ease of use

In User Story A.2.3, a need for separation between accelerator sections is mentioned. As seen in Figure 6, the overview for these sections in the BIS-streams GUI is contained in a single view. This is not optimal, as operators want to have different screens related to different accelerator sections. A mockup was created, as seen in Figure 11, to fulfill this need.

User Story A.5.7 mentions a need for a cyclic accelerator overview, which the BIS-streams GUI lacks. It is available in BIS-monitoring, as seen in Figure 5. This is also addressed by the mockup in Figure 11.

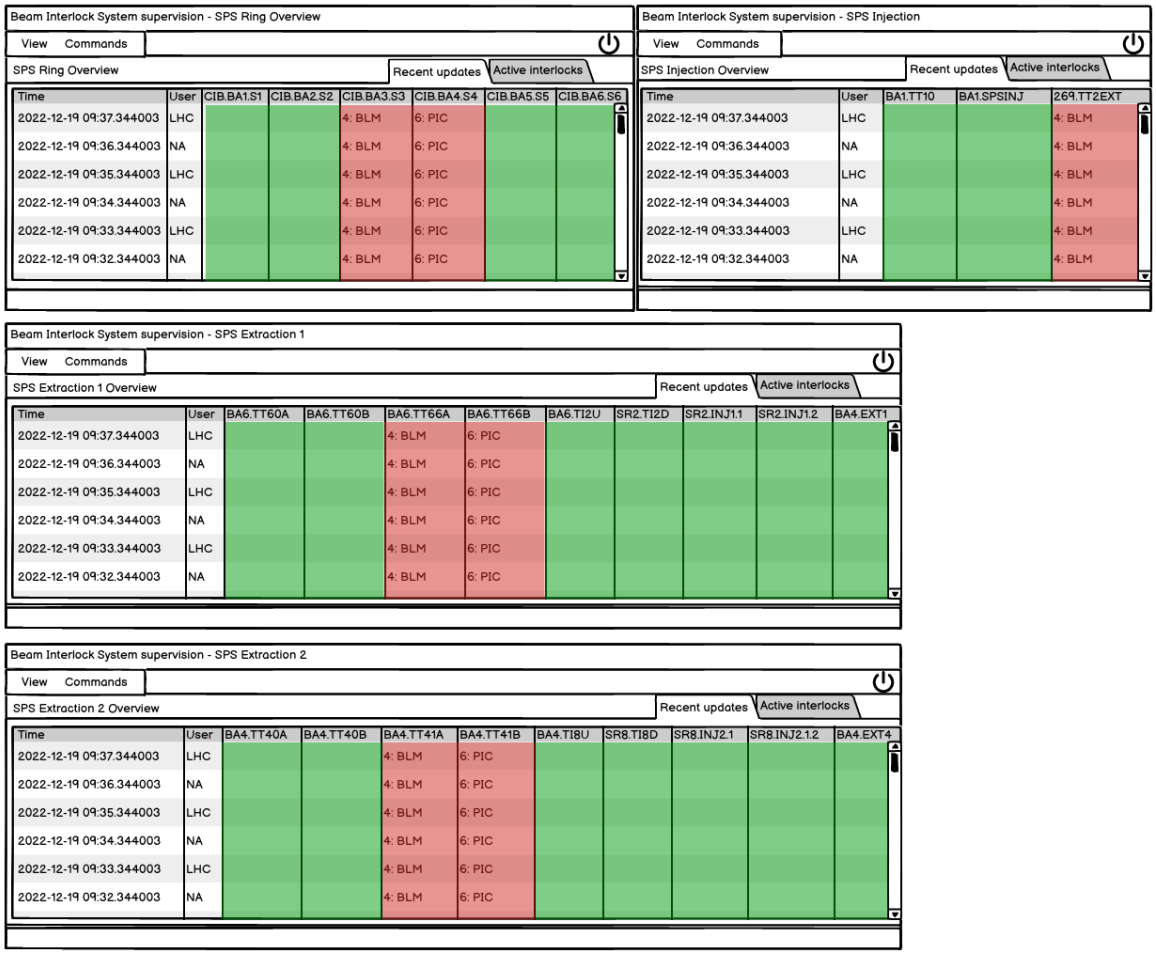


Figure 11: Mockups of the SPS section overviews.

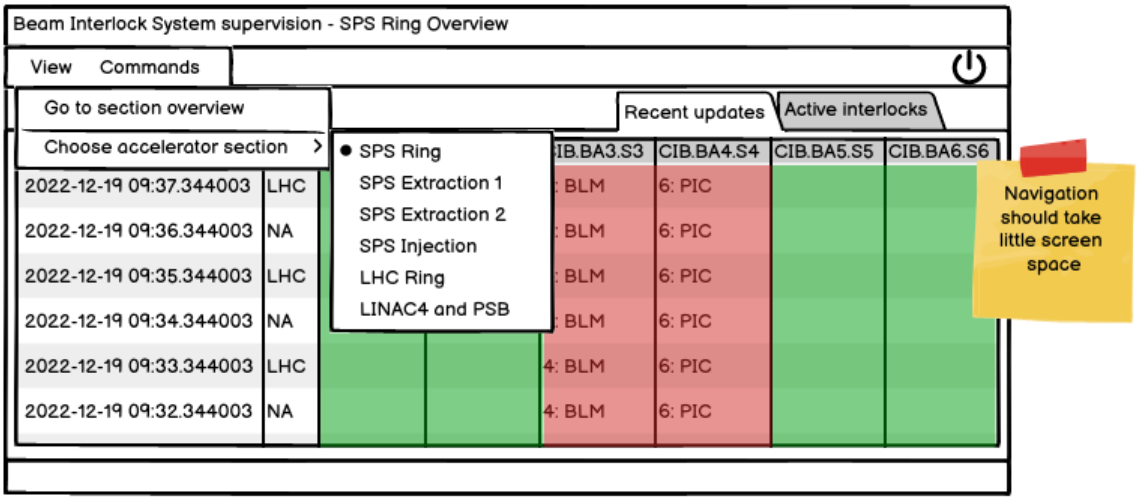


Figure 12: Mockup of changing accelerator context.

User Story A.5.2 describes a need for an overview of the interlocks in an accelerator, which can be seen in Figure 13.

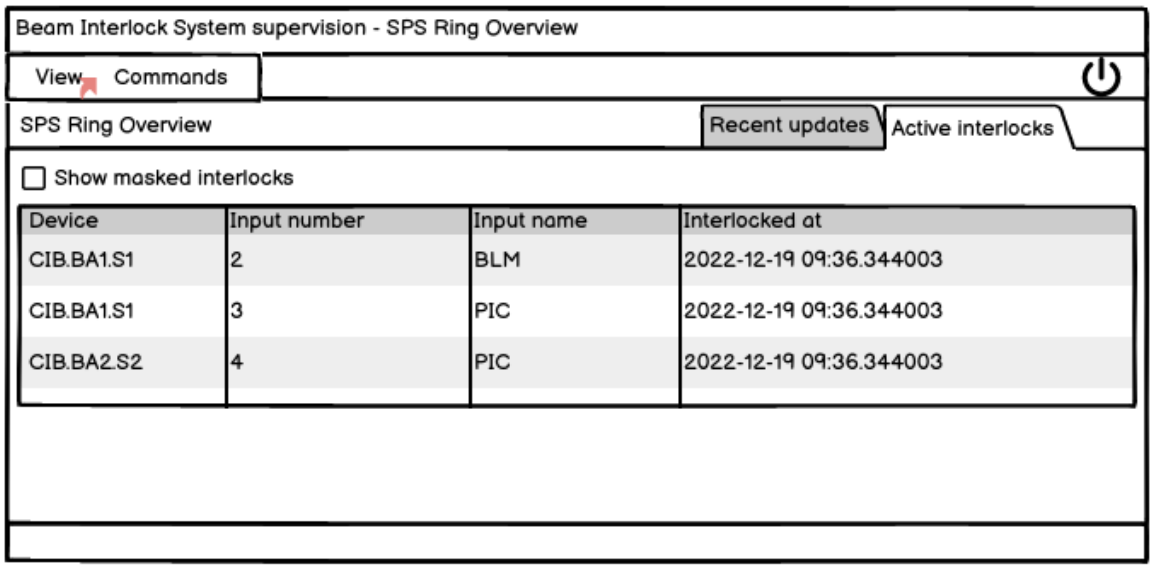


Figure 13: Mockup of active interlocks view.

5.3 Feedback

Two sessions were held to get feedback on the mockups from experts and operators respectively.

5.3.1 Expert

In terms of the GUI, the experts did not have a lot of feedback. The main issue was the navigation. Experts want to easily access devices and understand which devices belong to which accelerator. A suggestion was created by Ivan Romera Ramirez for navigating for devices with a tree view, as seen in Figure 14. The issue with this suggestion is that it interferes with User Story A.4.2, in which operators need compact information and have limited screen space. An alternative to this was suggested, where navigation to the different accelerators is done via the menu bar, as seen in Figure 12. The devices can then be seen in the recent updates view, and the device names should be clickable.

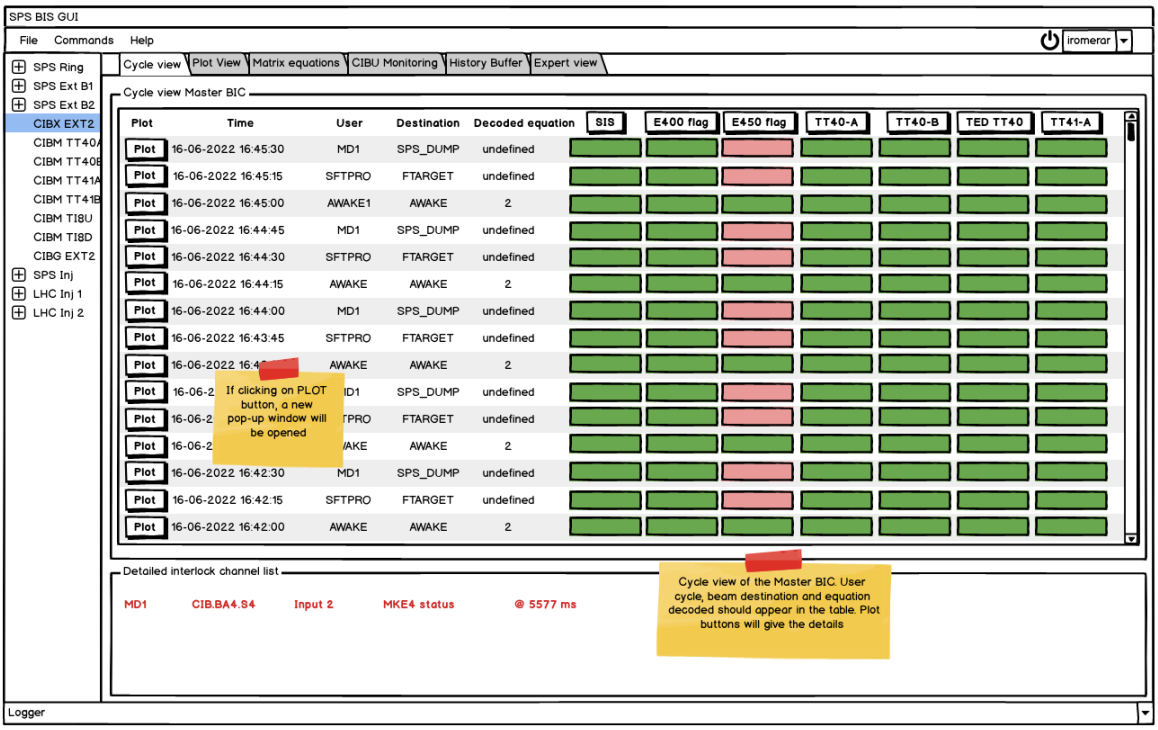


Figure 14: A proposed layout for the recent updates view, with a tree view navigation for accessing devices on the left. ©CERN, Ivan Romera Ramirez

5.3.2 Operator

From the operator point of view, BIS-UCAP was also interesting. The main focus for the discussion was the SPS use case, as seen in Figure 11.

One issue that appeared in the discussion was a confusion in naming conventions. A User Permit for an operator may refer to the experiments, since they are the users from operations' point of view. What experts call User Permits, operators refer to as input permits.

In terms of feedback for the mockups, Figure 13 is missing the user that interlocked. For the recent updates in Figure 11, it was requested that interlocking inputs for a single device should be listed in the order they are interlocked, so that an operator may get an immediate overview over the machine without having to do deeper investigation. Furthermore, a filter for specific users was requested for the recent updates view.

6 Implementations

This section covers how the different proof of concepts were created, some of their shortcomings and their architecture. The proof of concepts displays a view that shows the current active interlocks and what inputs that are causing them. In the case of fixed displays, device information is displayed, since active interlocks information is not directly provided by BIS-UCAP yet.

6.1 ACW

The Accsoft Commons Web (see Section 3.6.2) implementation was based on a boilerplate project provided by the BE-CSS group. The boilerplate had not been updated in quite a while, so there were some issues with outdated dependencies.

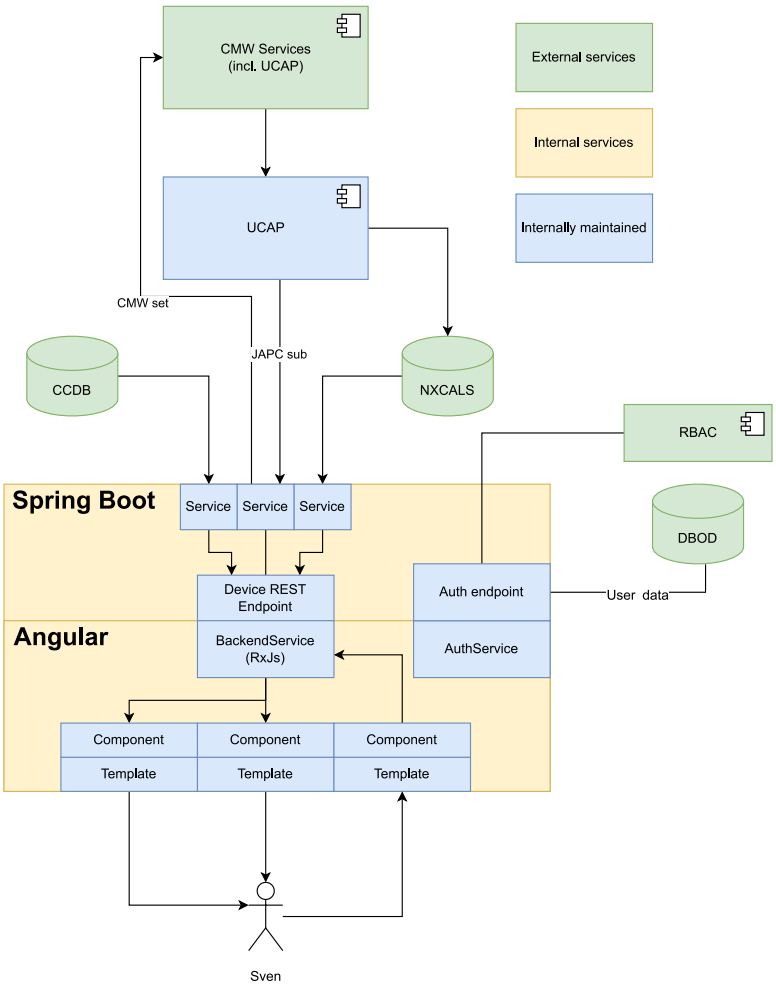


Figure 15: The architecture for implementing the ACW framework. A Spring Boot backend is used to fetch data from CERN services, and expose them to the Angular frontend. For the proof of concept, a REST endpoint was used, but especially for device subscriptions, WebSockets would be preferred. Hosting for the backend and frontend is required, which is displayed with the yellow color.

The backend based on Spring Boot provides a REST endpoint for getting device data. Data is being fetched at a set interval from the Angular frontend. This method was used instead of WebSocket subscriptions because of lesser implementation overhead.

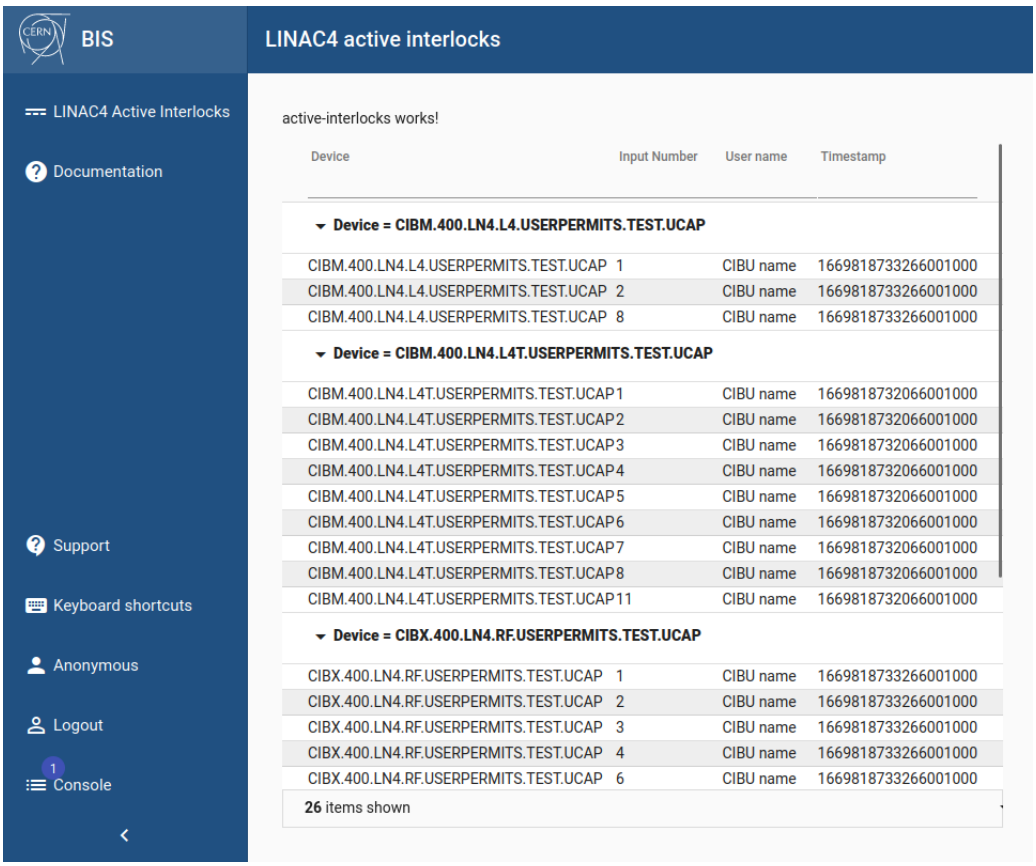


Figure 16: ACW proof of concept application displaying LINAC4 active interlocks in a table.

The implemented proof of concept can be seen in Figure 16. Pre-built components from the ACW framework were used, such as the acw-navigator (which provides the layout for the blue areas) and acw-table (which is used to display the currently interlocking devices). acw-table proved to be very feature rich, providing features such as grouping and filtering out of the box. The framework also provides a logging service that logs messages to the in-application console, which makes troubleshooting easy.

The boilerplate project was not adapted to be run on the Technical Network (TN), which does not have internet access. The virtual machines provided by the CERN IT department runs CentOS7, which provides an outdated version of Node.js. This makes the project very hard to set up on these VMs. In order to not have to install Node.js anew when setting up the development environment, a Docker image was created to run the frontend development environment. This image set a custom npm repository that points to CERN's Artifactory that hosts a proxy for the npm repository. There were also issues with Gradle trying to fetch dependencies, but this never needed to be resolved, as the UCAP TEST nodes were exposed to the General Purpose Network (GPN), which means that development could be done on office PCs with internet access, rather than the VMs.

6.2 PyQt

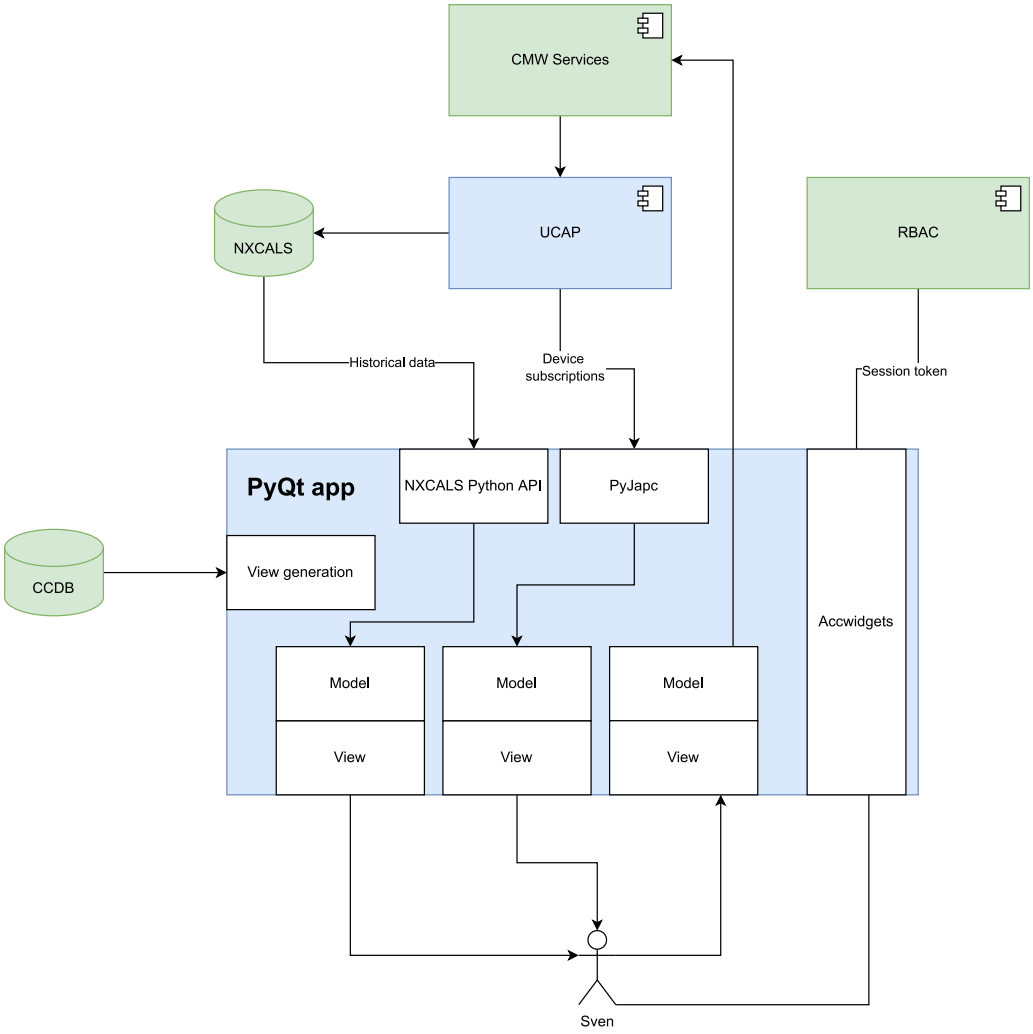


Figure 17: Architecture suggestion for the PyQt application. By using libraries provided via acc-py, most code can be contained within a single application.

A suggestion for an architecture based on PyQt was created, as seen in Figure 17. For the proof of concept, only the subscription data flow, using PyJapc, was implemented. PyJapc is a Python wrapper around JAPC. This was used to display a list of active interlocks from UCAP.

The PyQt proof of concept was built with accwidgets' ApplicationFrame. It integrates a console that prints logging statements, a button for RBAC authentication and a widget area as the body of the application. The solution can be seen in Figure 18.

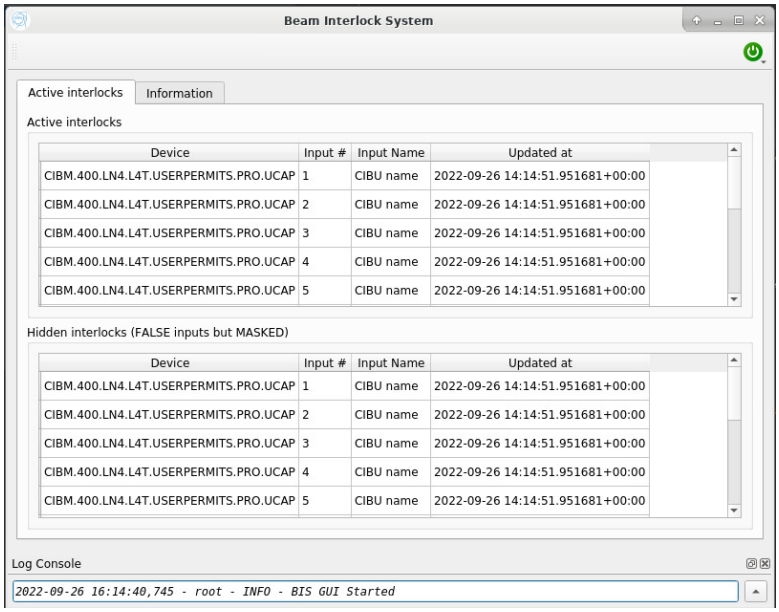


Figure 18: Python-based proof of concept. The main view contains two tables with active interlocks. The two tables display the same data, and the titles are only there for demonstration purposes.

The main widget consists of a tab with a content area. In this area, there are tabs that may be used to display different views. The only implemented view is one with two tables that display active interlocks.

6.3 Grafana

A proposed architecture for how device data could be visualized in Grafana was created, as seen in Figure 19. This includes a custom data source for querying device information. A proof of concept, using the same REST API as for ACW, with a different architecture can be seen in Figure 22. A set of views were also created, in order to compare functionality with existing GUIs. These were populated with data from the BIS-UCAP pipelines.

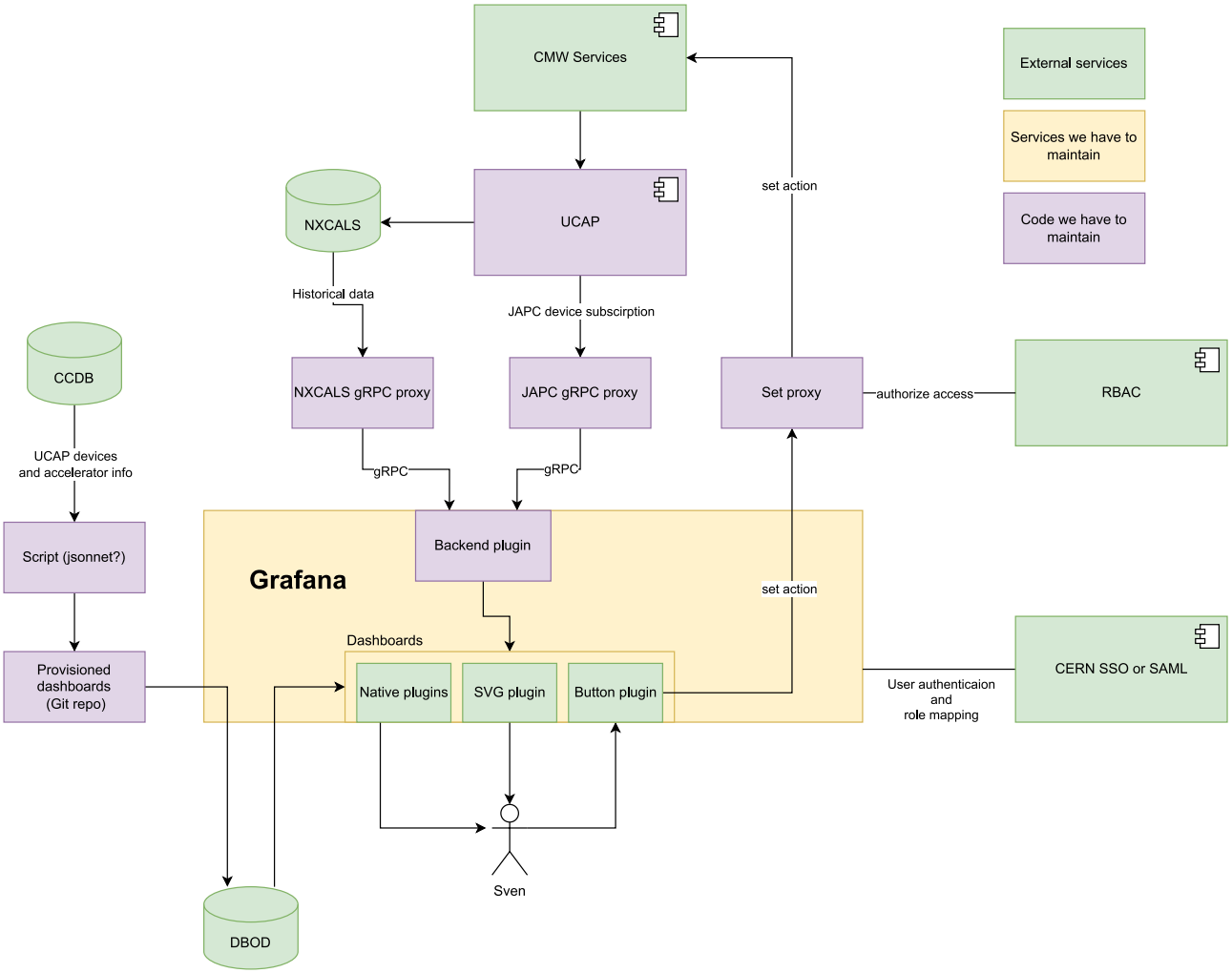
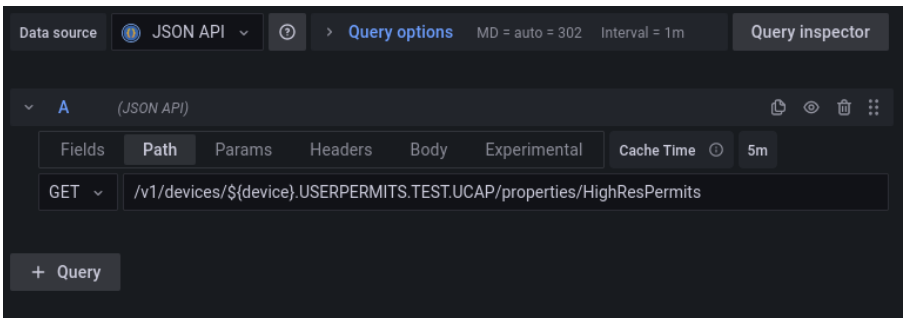
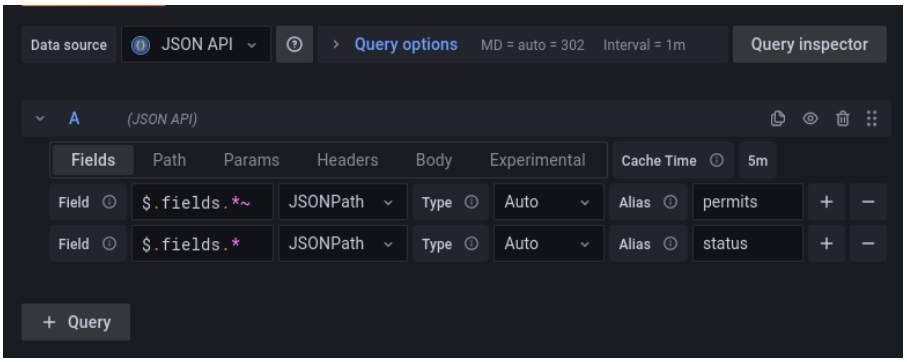


Figure 19: Architecture suggestion for the Grafana solution. A proxy is needed between the CERN services using the NXCALS extraction API and JAPC, as well as developing a backend plugin for Grafana. Dashboard creation and maintenance can be managed using scripting and Grafana's dashboard provisioning. Authentication can be integrated directly with CERN's Single Sign-On solution in the Grafana configuration.

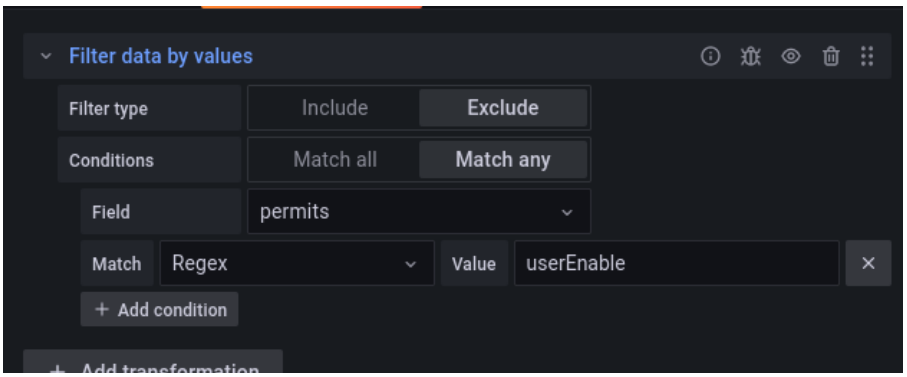
For the proof of concept, an alternative data source was used, as seen in Figure 22. The JSON API data source makes it possible to query an endpoint that returns any JSON (Figure 20a). The data from the JSON-file can then be read by Grafana using JSONPath or JSONata, as seen in Figure 20b. In some cases this data needs to be transformed, as seen in Figure 20c, where certain values are filtered out.



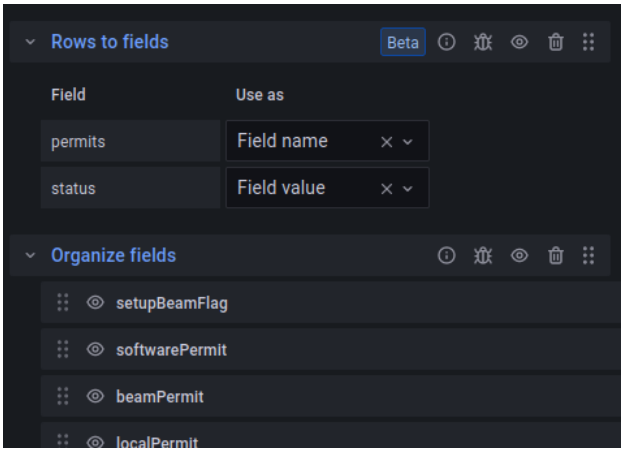
(a) The API path configured for the JSON API data source. A wild card for device name is used to leverage Grafana's template variables. This queries the high resolution user permits.



(b) The JSONPath queries in order to decode the JSON for Grafana fields. Here, the key-value pair is assigned two different fields; one called permits, and one called status.



(c) For the case of high resolution user permits, the *userEnable* fields were filtered out using a Grafana transformation, as this information is also provided by another source.



(d) The *Rows to fields* transformation was used in order to turn the property fields into Grafana fields. With the rows as fields, the fields can be ordered using drag-and-drop with the *Organize fields* transformation

Figure 20: Panel query configuration for the JSON API data source

Figure 21: CIBM information displayed in a Grafana dashboard with a set of Stats panels. The JSON API data source is used to fetch live information from UCAP, via the REST endpoint provided by the ACW proof of concept.

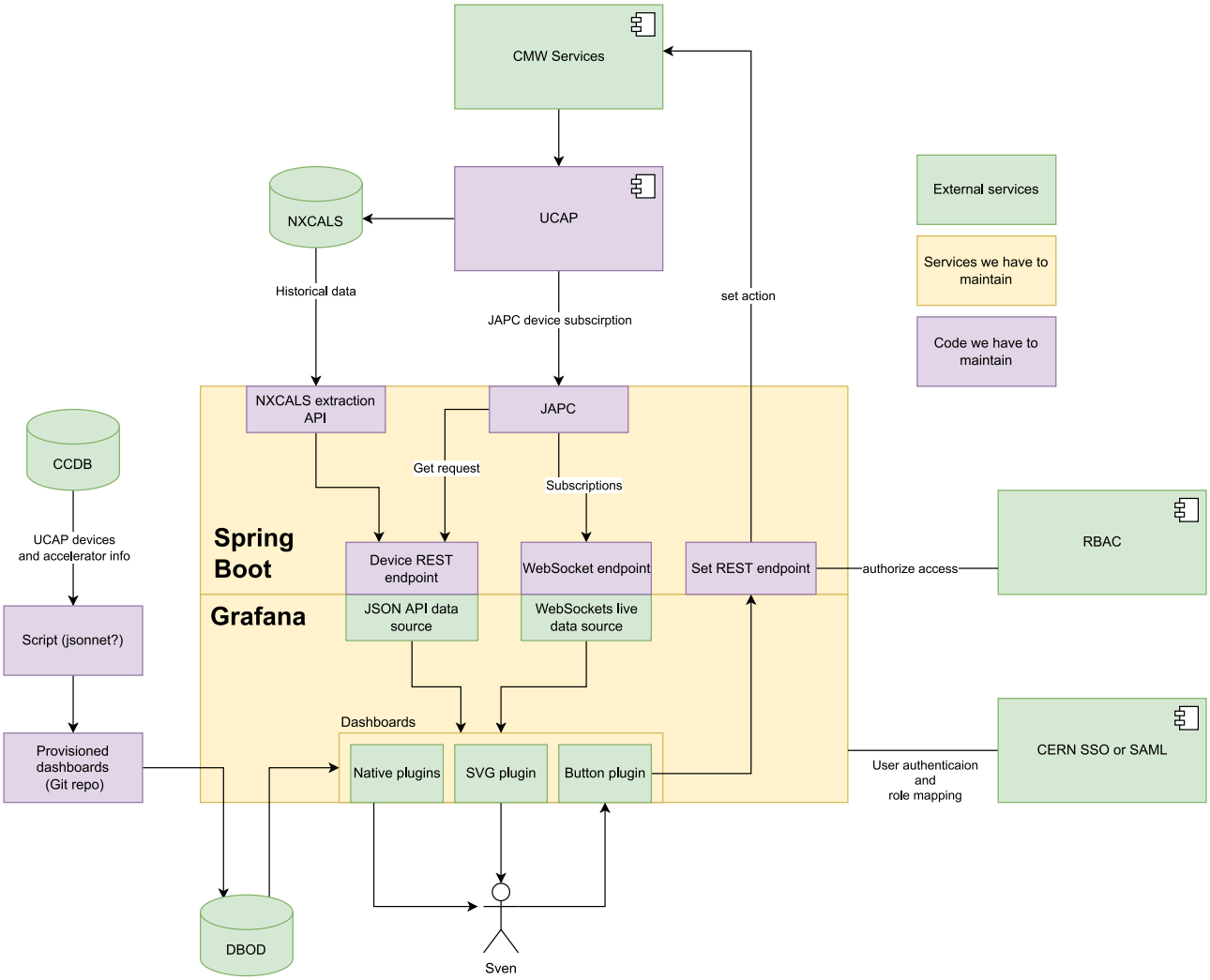


Figure 22: Architecture suggestion for the Grafana proof of concept. Only the device REST endpoint was implemented, the same as in the proof of concept for ACW seen in Figure 15. Instead of using a custom data source as in Figure 19, two open source alternatives are proposed; JSON API data source [25] and WebSocket API data source [26]

In order to implement a device data source, a set of domain knowledge for Go and Grafana plugins is required. This competence cannot be guaranteed to be available for the team during the operational time of the UI. Implementing a plugin in Java that interacts with Grafana's plugin API via gRPC could be an option, however this increases the complexity of development and deployment.

Plugins for SVG visualization [27] and API interaction [28] already exist, something which would minimize the amount of frontend code needed for development. These are third party plugins, so some extra consideration might be needed before using them.

Grafana is developing a Canvas panel, which provides similar functionality to the WRAP dashboard building, with simple drag-and-drop functionality. [29]

At the time of writing, Grafana lacks support for microsecond timestamps. [30] An example dashboard was created in order to validate this issue. [31]

6.4 WRAP

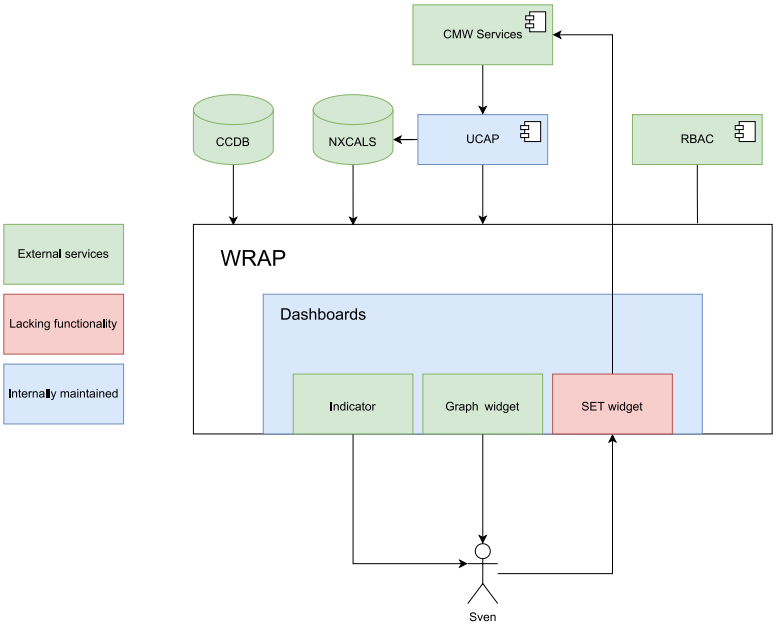


Figure 23: Architecture for WRAP. It is tightly integrated with the CMW systems, so no extra services need to be maintained. The set functionality is too limited for the requirements.

WRAP provides good integration with the accelerator control system. However, in terms of creating scalable dashboards and applications, it is a bit lacking.

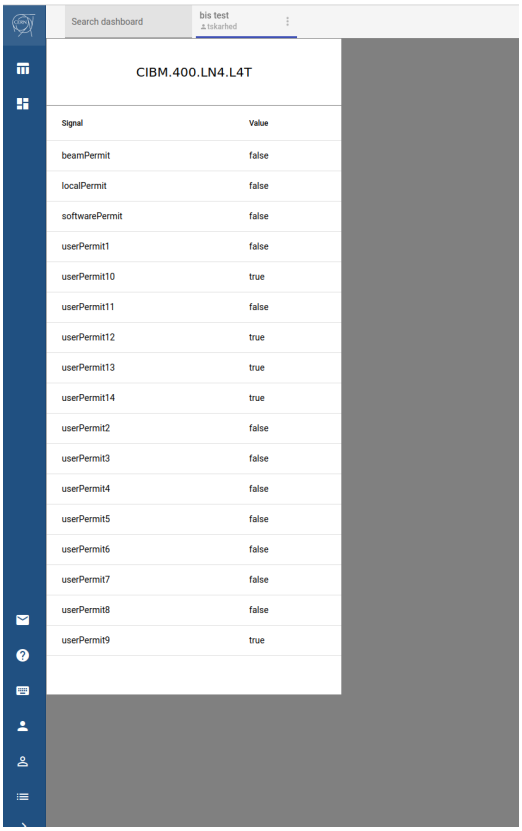


Figure 24: Dashboard in WRAP for displaying user permits for a single device. A label widget is used for the device name. A CMW Subscription Data Grid widget is used to display the user permits.

While WRAP provides variables for devices, it proves to not be enough for the BIS-UCAP use case. The implemented dashboard in Figure 24 is not dynamic, and only works for the single device. This is because the

templating system only works with full device names rather than string replacement. The way BIS-UCAP data is published is one device per feature, with the feature name being appended. If the device name would be available for string replacement, multiple feature could be available on the same dashboard.

The variable device name is not accessible for the widget component either, so the name is hard coded.

In terms of displaying user permits, all other solutions offer the possibility to map a color to the value, while this is not something available in the CMW Subscription Data Grid widget. Hence, it may be difficult for operators to quickly identify changes in user permits or interlocks.

The displayed user permits are not in order, which is an issue with any solution that reads from JAPC. In the Grafana case, it was possible to reorder them using transformations, something which WRAP lacks.

7 Discussion

This section discusses the method and useful it was, the evaluation of the mockups and technical limitations of the GUI technologies that were not directly reflected in the result. The thesis is concluded by a list of observations of the things discussed. Furthermore, ethical aspects are considered and areas for future studies are presented.

7.1 Method

Using user stories to compare and score different technologies provides a good overview, it does however lack in-depth analysis that other methods provide. Since it is highly dependent on what stories are chosen for scoring, it may be hard to reproduce and personal biases may come into play. Given the context, scope and purpose of the evaluation, the method fit well.

KAMP would have served better for a more formal comparison, where similar change requests could be compared across architectures by the length of task lists provided by change propagation throughout the system. As previously mentioned, KAMP was unsuitable for other reasons, such as lack of documentation, complexity and lack of focus on stakeholders.

The technological constraint of nanosecond timestamp support, User Story A.4.1, for web solutions could not have been discovered without previous domain experience. This applies to both experience with web technologies and experience with the CMW systems and their provided nanosecond timestamp. User stories as a methodology could not have guaranteed to identify this issue, hence identifying one of the method's flaws: low probability of reproducing the results. This is a similar drawback that can be found in the scenario based architecture analysis methods.

Even with the lack of reproducibility, the user stories provide more value compared to KAMP, in the sense that they may be used extensively for further work.

KAMP may be more useful for domains where changes are more expensive, such as hardware developments or production lines.

7.2 Mockup evaluation

The resulting mockups served little purpose in terms of choosing and evaluating technologies. It did however prove to be a good starting point for discussing users' needs.

The direct feedback from the expert point of view was not very extensive. This may be an indicator that the currently available UI serves their use case to a greater extent, compared to the operator point of view.

The scope of the mockups was also very limited, which may be a reason for the feedback not being very extensive.

For the operator use case, the feedback was more extensive. This may be because they have a greater need for having as much information as possible visible n the same view. The discussion with operations was very productive, as some new ways of displaying information were presented.

7.3 Technical implementations

A lot of the user stories were related to user experience and features that are needed, most of which were technology-agnostic. These were the stories that directly affected choice of technology:

- User Story A.4.1, defining the need for microsecond timestamp support
- User Story A.4.2, describing the need for compact information that may fit with other things on the screen
- User Story A.4.3, defining the need for decoupling windows and frames

In the case of User Story A.4.3, the web is a natural choice, as pages are meant to move around. In a similar sense, it proves a limitation for the compact information required by User Story A.4.2, because of the browser's navigation bar. This issue is possible to circumvent using a web-desktop solution such as Electron [32] or Tauri [33]. This issue is not present with the PyQt implementation. However, handling multiple windows with PyQt needs consideration during further development.

7.3.1 Limitations of fixed monitor solutions

WRAP and Grafana are both fixed-monitor solutions. This means that further modification is needed in order to handle set actions, which is one of the themes identified for the user stories (Appendix A.1). Both of them are also web-based. This produces some constraints in terms of timestamps. The native browser DateTime implementation does not support microsecond or nanosecond timestamps, as required by User Story A.4.1. This means that further modifications are needed to both WRAP and Grafana in order to be able to accurately graph microsecond data points. These issues can be solved, but it has to be done by third parties.

Fixed display solutions provide a very good separation of concerns, so no shortcut transformation may be implemented in the UI. Data transformations are supposed to live in BIS-UCAP, and by having this hard separation of concerns, it forces the developer to do all transformations where they are supposed to live. Having the transformations in BIS-UCAP also creates a public API for other people to consume.

Grafana provides a caveat to the hard separation of concerns, as transformations are available on the client side. These transformations are very limited, and may actually be good for last step filtering and reordering of data.

7.3.2 Architecture complexity

The clear difference between each architecture suggestion (Figure 9, Figure 15, Figure 17, Figure 19) is the amount of multiple services that needs to be managed and maintained. WRAP has the advantage that nothing needs to be hosted. PyQt is deployed as a single program without external services. ACW and Grafana require hosting services to a greater extent. In the case of Grafana, it is possible to exclude the dependency on dashboards as code and DBOD, by just using the default database and built in version management system.

The added complexity to ACW and Grafana may not necessarily be a bad thing. For a single team maintaining the services, there may be some more overhead. The JAPC proxy displays a need that is not yet provided as a service. More systems than the BIS need device subscriptions and set action through a web based API, especially if ACW as a framework wants wider acceptance at CERN.

As the user stories in Appendix A.1 describes several types of actions required by users, having a JAPC set action proxy may not be good enough. An API endpoint for the actions may need to implement specific behavior in order to do batch settings etc. of devices, both on an accelerator basis and on a device type basis. In this case, a simple JAPC proxy may not be enough, as a service that implements these actions would fulfill the requirements in a better way.

The two different Grafana architectures (Figure 19 and (Figure 22) may be useful for different purposes. Figure 19 describes a system where a Grafana plugin needs to be maintained. This would make it easy for any user to create pure JAPC queries in the Grafana UI. From a usability standpoint, this is preferred. This solution requires deeper knowledge about Grafana and its plugin APIs, which may not be suitable for a smaller team. If a team at CERN were to provide Grafana as a service, the responsibility for maintaining such a plugin would be theirs, which is a good scope. For this to be possible, a higher level strategy for using Grafana with the control system would need to be accepted.

Figure 22 describes a less domain specific approach. In this case, users who create dashboards need to be familiar with the underlying web-based API, rather than the JAPC concepts. Some knowledge about JSONPath and JSONata is needed as well. This increases the difficulty to onboard new users. For this approach, way less code needs to be maintained, as there is a Grafana plugin that needs maintenance.

7.3.3 API stability

The web is a rapidly changing platform, which means any web solution require constant maintenance and upgrades. [4] In terms of WRAP, this is not an issue for TE-MPE-CB, as it is maintained by BE-CSS. Grafana releases a new major version every year [34], with no long term support. This may impact the plugin APIs with breaking changes. When implementing the ACW proof of concept, a significant hurdle was upgrading the Angular dependencies to the correct version, something which needs to be done on a regular basis.

Another issue noticed while implementing the ACW proof of concept was that the API for the table component was changed without a major version bump. This is not according to Semantic Versioning (SemVer), as breaking changes requires a major version bump. It may cause issues with upgrades in the future if SemVer is not followed.

7.4 Conclusions

Overall, the initial question of which GUI technology is best for the next version of the BIS supervision GUI has been answered. The areas for evaluation may be summarized in this manner:

- Python Qt (PyQt) is the best choice of technology, as it scored the higghest and fulfilled the hard requirements
- Choice of technology does not have big impact on the ability to implement wireframes
- Wireframes as a tool is not good for evaluating technologies
- Fixed display solutions contain technical constraints that may increase complexity
- WRAP is too simple for using as a control application

The method used was appropriate for the scope of the evaluation and the purpose of providing a base for further work.

7.5 Sustainability and ethical implications

Decision makers need to do be well-informed about the different technologies that may be used and what long term effect it will have. This may include managing complexity of the technology as a source for developer stress level, or making it harder to onboard new people. Future extendability and maintenance of these technologies are also important to ensure longevity of the product. Therefore, it is important to evaluate the different options.

However, evaluations do not come without a cost. By choosing a method that is very theoretical, such as those listed in Section 2.2, one may end up in analysis paralysis. Outcomes may be very abstract, and the time spent may have limited influence on future development. This is an aspect that also needs to be considered. How in-depth evaluation can an organization afford, and how much value will it provide? These also seem to counter recent agile development methodologies, where delivering software to the end users often and quickly are a priority.

Sustainability-wise evaluations are important to ensure a good developer experience and software longevity. If the method chosen for an evaluation is not in line with an organization's goals and available resource, it may be considered harmful, which is an ethical issue.

7.6 Future work

The field of observability is interesting for cloud native applications. With metrics, logs and traces it is easier to troubleshoot applications. In the context of control applications, these things are still relevant, but another layer for acting on the insights should be added. In terms of Grafana, this can be done with the help of plugins. An official plugin from Grafana or an extension to it would be helpful. WRAP has no way of doing this to the required extent, but since they work closely with their stakeholders, it should only be a matter of time before functionality is extended.

The BIS-UCAP project needs to be extended to include accelerator-based APIs that aggregate data from devices for a specific accelerator. This is done to some extent with BIS-monitoring's UCAP project, as seen in Figure 8 and in User Story A.5.2. However, it was designed with the LHC in mind, so some investigation is needed to understand how it may be applied to other accelerators, especially the cyclic ones.

The user stories in Appendix A will serve as a basis for further development of the BIS-UCAP API and the user interface that consumes it.

With a chosen technology, the created architecture diagrams may be further developed for more low level class diagrams. The methods in Section 2.2 may be useful to evaluate and improve these architectures.

The upcoming focus on the BIS UI will focus on further iterations of creating wireframes/mockups. The next step will be to formally decide the most appropriate technology, something this thesis may be used for.

References

[1] Roland Johnson et al. "The Consolidation of the CERN Beam Interlock System. THE CONSOLIDATION OF THE CERN BEAM INTERLOCK SYSTEM". In: *JACoW IPAC* '21 (2021), pp. 3309–3312. DOI: 10.18429/JACoW-IPAC2021-WEPAB282. URL: http://cds.cern.ch/record/2810349.

[2] Marc-Antoine Galilée et al. "Renovation and extension of supervision software leveraging reactive streams". In: *ICALEPCS2017*. 2018, THPHA152. 4 p. DOI: 10.18429/JACoW-ICALEPCS2017-THPHA152. URL: http://cds.cern.ch/record/2305520.

[3] Aleksander Buszydlik. "Extending the Control Software for Beam Interlock System 2". CERN Summer Student Programme Final Report. 2022. URL: http://cds.cern.ch/record/2826590/.

[4] Ivan Sinkarenko, Vito Baggiolini, and Sara Zanzottera. "Our Journey from Java to PyQt and Web for CERN Accelerator Control GUIs". In: (2020), TUCPR03. DOI: 10.18429/JACoW-ICALEPCS2019-TUCPR03. URL: http://cds.cern.ch/record/2779634.

[5] Zsolt Kovari and Grzegorz Kruk. "New Timing Sequencer Application in Python with Qt - Development Workflow and Lessons Learnt". In: *JACoW* ICALEPCS2021 (2022), 904–907. 4 p. DOI: 10.18429/JACoW-ICALEPCS2021-THPV015. URL: http://cds.cern.ch/record/2809472.

[6] Evgeny Alexandrov, Giuseppe Avolio, and Eukeni Pozo Astigarraga. *Usage of Time Series Databases in the Grafana Platform for the Netis Service*. Tech. rep. Geneva: CERN, 2021. URL: http://cds.cern.ch/record/2781405.

[7] Muhammad Ali Babar and Ian Gorton. "Comparison of scenario-based software architecture evaluation methods". In: Jan. 2004, pp. 600–607. ISBN: 0-7695-2245-9. DOI: 10.1109/APSEC.2004.38.

[8] Kiana Rostami et al. "Architecture-Based Assessment and Planning of Change Requests". In: *Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures*. QoSA '15. Montréal, QC, Canada: Association for Computing Machinery, 2015, 21–30. ISBN: 9781450334709. DOI: 10.1145/2737182.2737198. URL: https://doi.org/10.1145/2737182.2737198.

[9] Stephane Deghaye and Eve Fortescue-Beck. *Introduction to the BE-CO Control System*. CERN, 2020. URL: https://cds.cern.ch/record/2748122.

[10] Stéphane Deghaye and Eve Fortescue-Beck. "CMW-RDAs". In: *Introduction to the BE-CO Control System*. CERN, 2020. Chap. 14.1.

[11] Stéphane Deghaye and Eve Fortescue-Beck. "JAPC". In: *Introduction to the BE-CO Control System*. CERN, 2020. Chap. 14.3.

[12] Lajos Cseppentő and Mark Büttner. "UCAP: A Framework for Accelerator Controls Data Processing @ CERN". In: *JACoW* ICALEPCS2021 (2022), 230–235. 6 p. DOI: 10.18429/JACoW-ICALEPCS2021-MOPV039. URL: https://cds.cern.ch/record/2809575.

[13] B Puccio et al. "The CERN Beam Interlock System: Principle and Operational Experience". In: *IPAC2010*. 2010, 4 p. URL: https://cds.cern.ch/record/1277643.

[14] Stéphane Deghaye and Eve Fortescue-Beck. "RBAC". In: *Introduction to the BE-CO Control System*. CERN, 2020. Chap. 13.2.

[15] "NXCALS - Architecture And Challenges Of The Next CERN Accelerator Logging Service". In: *17th Int. Conf. on Acc. and Large Exp. Physics Control Systems*. JACoW Publishing, 2019. DOI: 10.18429/JACoW-ICALEPCS2019-WEPHA163.

[16] *CERN's OpenShift instance*. URL: https://paas.docs.cern.ch/ (visited on 02/23/2023).

[17] *Grafana on CERN's OpenShift*. URL: https://grafana.docs.cern.ch/ (visited on 02/23/2023).

[18] Stéphane Deghaye and Eve Fortescue-Beck. "Graphical Frameworks and Components". In: *Introduction to the BE-CO Control System*. CERN, 2020. Chap. 17.1.

[19] Epameinondas Galatas et al. "WRAP - A Web-Based Rapid Application Development Framework for CERN's Controls Infrastructure". In: *JACoW* ICALEPCS2021 (2022), 894–898. 5 p. DOI: 10.18429/JACoW-ICALEPCS2021-THPV013. URL: http://cds.cern.ch/record/2809470.

[20] *Stateful Scalar Vector Graphics Specification*. 2020. URL: https://mro-dev.web.cern.ch/docs/std/ssvg-specification.html (visited on 12/01/2022).

[21] *Grafana releases: New 2023 release schedule*. URL: https://grafana.com/grafana/plugins/ (visited on 02/23/2023).

[22] *Grafana plugin examples*. URL: https://github.com/grafana/grafana-plugin-examples (visited on 02/23/2023).

[23] Frank Locci et al. "CERN Controls Open Source Monitoring System". In: (2020), MOPHA085. 5 p. DOI: 10.18429/JACoW-ICALEPCS2019-MOPHA085. URL: https://cds.cern.ch/record/2778510.

[24] *Jsonnet*. URL: https://jsonnet.org/ (visited on 02/23/2023).

[25] *Grafana JSON datasource*. URL: https://grafana.com/grafana/plugins/marcusolsson-json-datasource/ (visited on 02/23/2023).

[26] *Grafana WebSocket datasource*. URL: https://grafana.com/grafana/plugins/golioth-websocket-datasource/ (visited on 02/23/2023).

[27] *Grafana ACE.SVG panel plugin*. URL: https://grafana.com/grafana/plugins/aceiot-svg-panel/ (visited on 02/23/2023).
[28] *Grafana Button panel plugin*. URL: https://grafana.com/grafana/plugins/cloudspout-button-panel/ (visited on 02/23/2023).
[29] *Grafana Canvas panel*. URL: https://grafana.com/docs/grafana/latest/panels-visualizations/visualizations/canvas/ (visited on 02/23/2023).
[30] *Grafana Feature request: Support nanosecond resolution*. URL: https://github.com/grafana/grafana/issues/6252 (visited on 02/23/2023).
[31] *Example dashboard verifying Grafana's lack of nanosecond timestamp support*. URL: https://gist.github.com/tskarhed/4f8dd61d67bf77b20ba1c6bb71684ce8 (visited on 02/23/2023).
[32] *ElectronJS*. URL: https://www.electronjs.org/ (visited on 02/23/2023).
[33] *Tauri*. URL: https://tauri.app/ (visited on 02/23/2023).
[34] Tony Leopold. *Grafana releases: New 2023 release schedule*. URL: https://grafana.com/blog/2022/12/13/grafana-releases-new-2023-release-schedule/ (visited on 02/23/2023).

A User Stories

These user stories were gathered by talking to stakeholders and evaluating functionalities in existing GUIs. Common themes were identified, and the stories were grouped accordingly.

A.1 Actions

These user stories cover reasons a user might need to perform an action on a system, based on the available data.

User Story A.1.1. *As operator and expert, I want to send test commands to the device inputs, so I can verify that it is working properly*

User Story A.1.2. *As an LHC operator, I want to be able to mask a set of subsystems, so I don't have to manually mask every input (MASK groups in BIS Monitor)*

User Story A.1.3. *As an LHC operator, I want to reset IR6 BPM interlocks, so that ???*

User Story A.1.4. *As an LHC operator, I want to reset RF interlocks, so that ???*

User Story A.1.5. *As an LHC/SPS operator, I want to set default Software permit timeouts in a certain context (ring, extraction line, etc.), so that BIS have properly configured software permits timeouts*

User Story A.1.6. *As an LHC/SPS operator, I want to set all SW to TRUE in a certain context (ring, extraction line, etc.), so that I don't have to set them individually for each controller board*

User Story A.1.7. *As an LHC operator, I want to re-arm all BICs in a certain context (ring, extraction line, etc.), so that operations can be resumed easily*

User Story A.1.8. *As an SPS operator, I want to re-arm all BICs in a certain context (ring, extraction line, etc.) after a system restart, so that operations can be resumed easily*

A.2 Development environment and technology

User Story A.2.1. *As an operator, I want to have compact information, so I can fit it with other programs on the monitor*

User Story A.2.2. *As an operator, I want to open things in a new window, so I can have an overview and details view at the same time*

User Story A.2.3. *As an SPS operator, I want to open subframes of the SPS main overview panel, so that I can configure my screens the way I want*

User Story A.2.4. *As a developer, I don't want to wrestle with constant manual dependency issues and updates, as it slows down the process of producing valuable code*

User Story A.2.5. *As a developer, I want a simple development environment, so I don't have to configure and set up a lot of things when getting started*

User Story A.2.6. *As a developer, I want code to be easily testable, so that bugs can be found quickly*

User Story A.2.7. *As a developer, I want to have a simple infrastructure to manage, so that I don't spend too much time with infrastructure maintenance and evolution*

User Story A.2.8. *As a supervisor, I want a set of technologies that the team is familiar with, so that I don't increase to complexity of the product portfolio*

User Story A.2.9. *As a supervisor, I want a technology with support from BE-CSS, so that issues can be resolved more easily and guarantee future long term support/compatibility*

User Story A.2.10. *As a developer, I want to provide documentation for users, so that I can reduce the support load*

User Story A.2.11. *As a developer, I want to provide documentation for API users, so that I can reduce the support load*

User Story A.2.12. *As a developer, I want to provide documentation for other developers, so that they can contribute easily to the project*

A.3 Device information

User Story A.3.1. *As an operator, I want to see inputs, masks and matrices of each individual board, so that I can troubleshoot a specific device*

User Story A.3.2. *As operator and expert, I want to view live updates to the history buffer, so I can understand how a device is operating*

User Story A.3.3. *As operator and expert, I want to view recorded data in NXCALS from the history buffer, so I can understand how a device has operated*

User Story A.3.4. *As an LHC operator, I want to see disabled channels, so I know which input values to rule out*

User Story A.3.5. *As an operator, I want to see how the permits change over time, so that I can understand the performance of my system*

User Story A.3.6. *As a cyclic machine operator, I want to plot all the permits during a cycle, so that I can see which permit interlocked first*

User Story A.3.7. *As a cyclic machine operator, I want to plot a single permit across multiple cycles, so that I can compare how it behaves cycle to cycle*

User Story A.3.8. *As a cyclic machine operator, I want to plot a single permit on a single cycle, so I can look into it into details*

User Story A.3.9. *As an SPS operator, I want to see which equation was applied for an interlock and which was expected, so I quickly understand any mismatch between the expected destination and the equation that enabled the beam, or interlocked it*

User Story A.3.10. *As an LHC operator, I want to read SIS timeouts, so that I can see which BIC is configured with timeout for the software permit and its value*

User Story A.3.11. *As expert, operator, SW engineer, I want to see the SW Permit Timeout configuration and current value in the expert panel, so that I know how it is configured and behaving*

A.4 Data acquisition and visualization

User Story A.4.1. *As expert, operator, SW engineer, I want microsecond time resolution, so I can verify the sequence of events correctly*

User Story A.4.2. *As an operator, I want to have compact information, so I can fit it with other programs on the monitor*

User Story A.4.3. *As an operator, I want to open things in a new window, so I can have an overview and details view at the same time*

User Story A.4.4. *As an operator, I want to see live permit updates in a graph, so I can understand the recent performance*

User Story A.4.5. *As a developer, I want a clear separation of concerns between UI updates and data management,to more easily track down bugs and make changes*

User Story A.4.6. *As an SPS operator, I want to have a custom state (different color than green or red) for the SBDS channel that is expected to be false at the end of the cycle, so that I know it is in the expected state*

User Story A.4.7. *As an SPS operator, I want to have a red color when the SBDS channel is not false at the end of the cycle, so that I know there is an error on this user permit*

A.5 Accelerator, injector and extractor overview

User Story A.5.1. *As an operator, I want to be able to see all masked interlocks, so that I know which system would interlock if the masks were removed*

User Story A.5.2. *As an operator, I want to see which system interlocked in an overview state, so I quickly can identify where the issue is*

User Story A.5.3. *As an operator, I want to see all devices in an accelerator overview, so I can quickly identify where an issue is*

User Story A.5.4. *As a system expert, I want to supervise any installation of the BIS, so that I can work with my system*

User Story A.5.5. *As an LHC operator, I want to see the current timing of the LHC, so that I know the conditions of the operations*

User Story A.5.6. *As an SPS operator, I want to be able to filter for USER for the accelerator updates, so that I can focus on the information I want*

User Story A.5.7. *As a cyclic machine operator, I want to have a cycle summary overview of the entire accelerator, so that I get a meaningful overview of the accelerator state*

User Story A.5.8. *As an operator, I want an integrated view that allows me to zoom in into issues, so that I get a meaningful overview and can navigate through what I want to investigate more deeply*

User Story A.5.9. *As an SPS operator, I want to have one screen per destination, so that the specific information between systems have the same context*