ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM007

DYNAMIC CONTROL ROOM INTERFACES FOR COMPLEX PARTICLE ACCELERATOR SYSTEMS

B. E. Bolling* , D. Nordt, G. Fedel, M. Munoz, European Spallation Source ERIC, Lund, Sweden

Abstract

The European Spallation Source (ESS) is a research facility under construction aiming to be the world's most powerful pulsed neutron source. It is powered by a complex particle accelerator designed to provide a 2.86 ms long proton pulse at 2 GeV with a repetition rate of 14 Hz. Commissioning of the first part of the accelerator has begun and the requirements on the control system interfaces varies greatly as progress is made and new systems are added. In this paper, three such applications are discussed in separate sections.

A Navigator interface was developed for the control room interfaces aimed towards giving operators and users a clear and structured way towards quickly finding the needed interface(s) they need. The construction of this interface is made automatically via a Python-based application and is built on applications in any directory structure both with and without developer interference (fully and semi-automatic methods).

The second interface discussed in this paper is the Operations Accelerator Synoptic interface, which uses a set of input lattices and system interface templates to construct configurable synoptic view of the systems in various sections and a controller panel for any selected system.

Lastly for this paper there is a configurable Radio Frequency Orchestration interface for Operations, which allows in-situ modification of the interface depending on which systems and components are selected.

INTRODUCTION

Complex machinery always poses multiple challenges when it comes to designing comprehensible graphical user interfaces (GUIs) such that the operators can work efficiently. Large-scale particle accelerators, with many unique subsystems are an extreme example of this. In this paper, the term Operator Interface (OPI) is used to describe any GUI that is used within the Phoebus framework [1] as it is intended to be used by an ESS Operator.

To address a few challenges of having to operate such complex machinery, two Python scripts were developed to dynamically build OPIs based on user inputs (accelerator lattice files). These OPIs are referred to as being dynamic as they are generated (updated) when needed. A third OPI is also described which is dynamic for the user during runtime.

OPERATOR INTERFACES

At ESS the OPIs are structured in three different levels aimed for control room operators ("Operator level"), system expert functions ("System Expert level") and for lowest-level functions ("Engineering level"). With this methodology implemented already during commissioning and conditioning phases, it became possible to find which settings the different levels of OPIs should be exposed to. Further on, each have a subset of directories for each part of the machine (e.g. Accelerator and Target), followed by another subset of directories for each system of that part of the machine (e.g. for the Ion Source section or Radio Frequency systems).

Within each machine-part directory lies the OPIs, which are designed in accordance with the internal OPI visual design rules document. The visual design document describes what colour codes are supposed to be used for what type of state and object, which type of object(s) to use for which type of setting/information, etc., in order to establish a site-wide continuity, with the outcome being a higher situational awareness for the control room operator and hence increasing the reliability of the facility [2].

NAVIGATOR OPI

The openness of Phoebus introduces the issue that having a large number of user interfaces scattered across a large amount of directories, with many cases having support files that cannot be opened as standalone applications, the user interfaces needed become difficult/time-consuming to be found. The OPIs Navigator OPI is a prototype interface aimed at solving the issue by dynamically constructing an OPI to enable users to navigate amongst relevant user interfaces - saving users a lot of time and effort. A user flowchart is shown in Fig. 1.

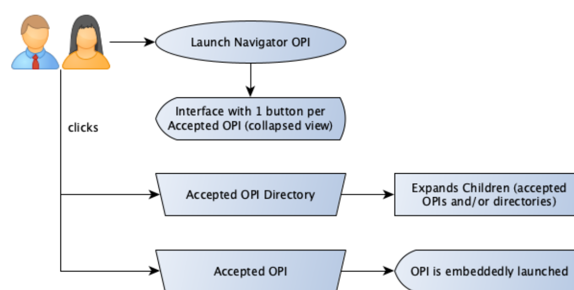

Figure 1: Navigator OPI user flowchart.

Methodology

A Python script is used to render the Navigator OPI which loops through a given set of directories using a recursive strategy for any subdirectory and with a set of filters applied, such as methods for identifying support files such that they can be omitted. The script then launches a PyQt5-based user interface with a table of OPIs identified such that the developer may select which OPIs are to be included in the

* benjamin.bolling@ess.eu

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM007

Navigator OPI, which may be referred to as accepted OPIs. A flowchart of the construction method can be seen in Fig. 2.

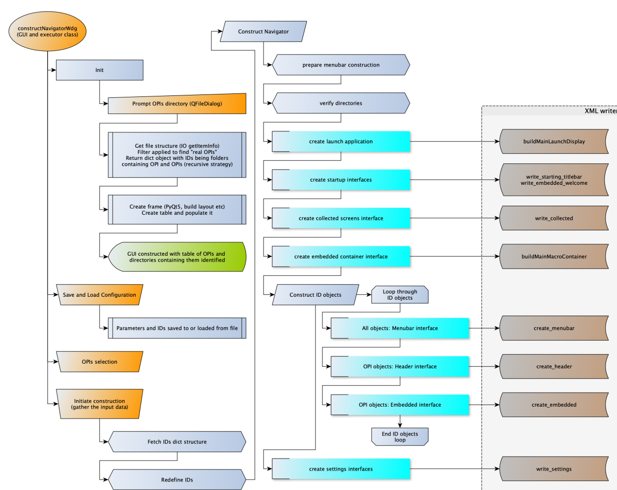

Figure 2: Navigator OPI construction method flowchart.

Each accepted OPI gets its own button and tree-structure leading to it showing all parent nodes (i.e. directories), and converts a file with the name *overview* or *main* setting the OPI to be its parent's button (see Fig. 3 showing Radio Frequency (RF) cavities' Overview OPI as the parent directory's name is RF).

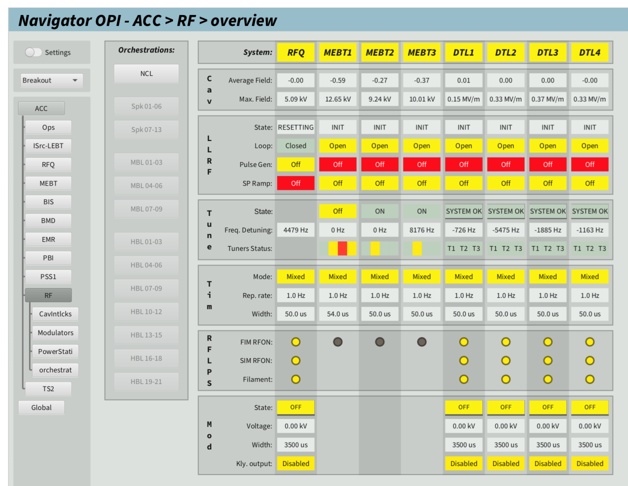

Figure 3: Navigator OPI showing the RF Overview OPI.

The Navigator OPI has been refined and readjusted multiple times during multiple ESS Normal Conducting Linac commissioning phases. The first version consisted of a single file with many objects listening to multiple other virtual signals, set up as so-called "rules" regarding the position of each button and their respective horizontal and vertical lines. This initial attempt was found to increase the internal memory usage exponentially as a function of the number of accepted OPIs with parents, making it unsustainable as the number of OPIs increased and is expected to increase further as the facility matures.

A better approach turned out to be to have a larger set of support files - one per OPI and with a single local signal listening to which OPI is the actively selected one. This method instead used a larger proportion of the hard disc (16MB for more than 300 OPIs), which with a linear increase of the space used as a function of the number of OPIs now considered as sustainable as the internal memory usage is reasonably fixed.

Results

The end result is a Python script dynamically constructing the Navigator OPI which enables fast and efficient navigation between accepted OPIs. There are further discussions ongoing on how to implement the Navigator OPI as a Phoebus element directly - hence it being referred to as a prototype - since it is now clearer what is needed from an Operations perspective with regards to an OPI Navigator.

ACCELERATOR SYNOPTIC OPI

The Accelerator Synoptic OPI, derived from the Greek word "synoptikos" which means general view of the whole, is designed to provide a general overview of the state of a linear accelerator as a whole combined with ways to quickly interact with each accelerator component. As with the Navigator OPI, it is constructed by activating a Python script with a set of embedded display templates, blockicons (pushbuttons combined with visual elements), filter setups, and one or more accelerator lattice files. A user flowchart of the application itself is shown in Fig. 4. The script sorts all components in terms of position along the accelerator, and builds the synoptic application with interactive elements and their associated embedded displays.

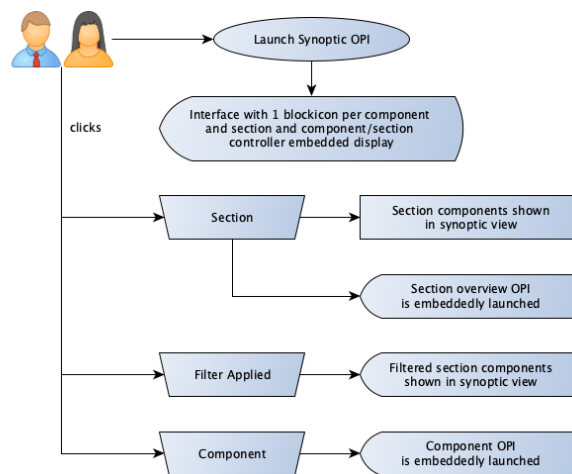

Figure 4: Synoptic OPI user flowchart.

Methodology

The initial version was created with similar methods as the initial version of the Navigator OPI, i.e. consisting of a

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM007

single file with many objects listening to multiple other virtual signals - resulting in that an unnecessarily large portion of the internal memory is used. Hence, the method behind how the application is built was refined such that the script instead creates one embedded display per accepted element and filter combination. An accepted element shall have a valid blockicon associated with it as well as a name of the physical equipment in the lattice file (as a lattice file can contain several "empty" elements, such as drift space, for e.g. accelerator physics applications). For each synoptic combination (show or hide component species, such as vacuum components), separate files are constructed and displayed as embedded screens per section (or combined section).

By using templates and different identifiers, the developer can easily modify and add or remove different components to information displays and the embedded displays (including controllers, readbacks, history plots, etc.). A simplified flowchart regarding how the Python script constructs the Synoptic OPI has been generated and is shown in Fig. 5.

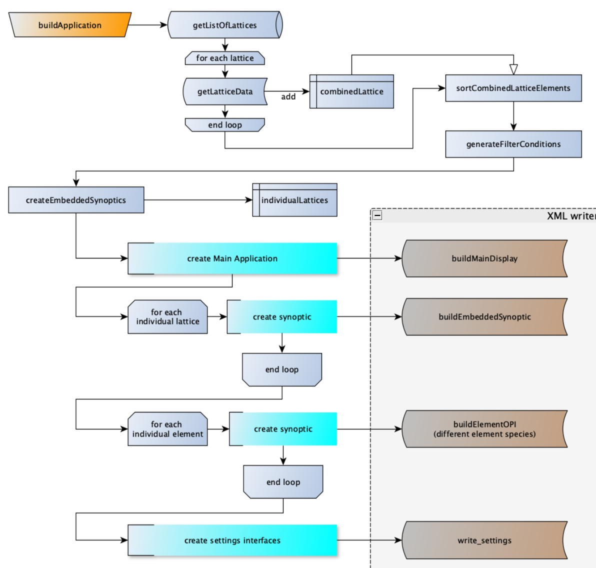

Figure 5: Synoptic OPI construction method flowchart.

Results

The end result is an Accelerator Synoptic OPI which is easy to use and enables users to quickly navigate through the different components as well as have an understanding of the overall state of the machine in accordance with the user flowchart in Fig. 4. Sectional overviews further adds to the completeness of the application, as can be seen in Fig. 6.

Furthermore, a filter function can be set up to show or hide different accelerator component species (see Fig. 7), interacting with/showing live data from any component (presuming that it had a template file enabling this) as well as information regarding it (with e.g. lattice data and a picture of it, see Fig. 8).

Figure 6: Synoptic OPI showing the Medium Energy Beta Transport sectional overview.

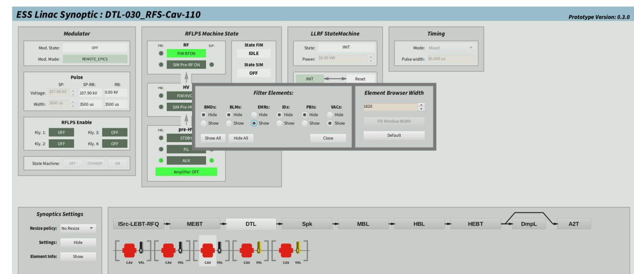

Figure 7: Synoptic OPI showing the accelerator component filter function and Drift Tube Linac section 3 RF controllers.

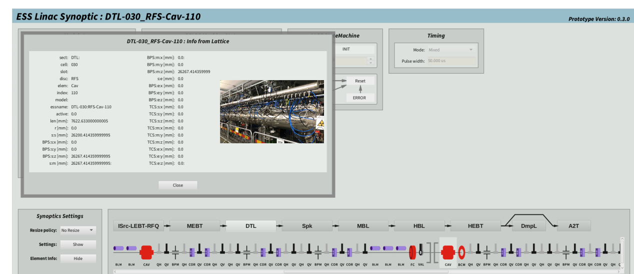

Figure 8: Synoptic OPI showing the Drift Tube Linac section 3 RF controllers and component information.

RF ORCHESTRATION OPI

The configurable RF Orchestration interface for Operations allows in-situ modification of the setup depending on which systems and components are to be controlled for different purposes. Using local signals, users can select which accelerator components (such as Medium Energy Beam Transport (MEBT) Buncher 1) to show and which subsystems (Modulator, Low-Level RF, etc.) to show detailed information about and controllers and for which to show only the essential information.

Methodology

The OPI consists of submodules with different inputs for their naming and can hence be reused for other similar systems. When combined, they span e.g. the RFQ and the DTL RF systems (2-klystrons-per-modulator system), meaning that a relatively small amount of work is needed to create the OPIs for different system species - such as solid state amplifier bunchers having no modulators or 4-klystrons-per-modulator systems.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM007

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

Results

A fully expanded view is shown in Fig. 9 and an almost fully collapsed view is shown in in Fig. 10 (in which only the Modulator module is expanded).

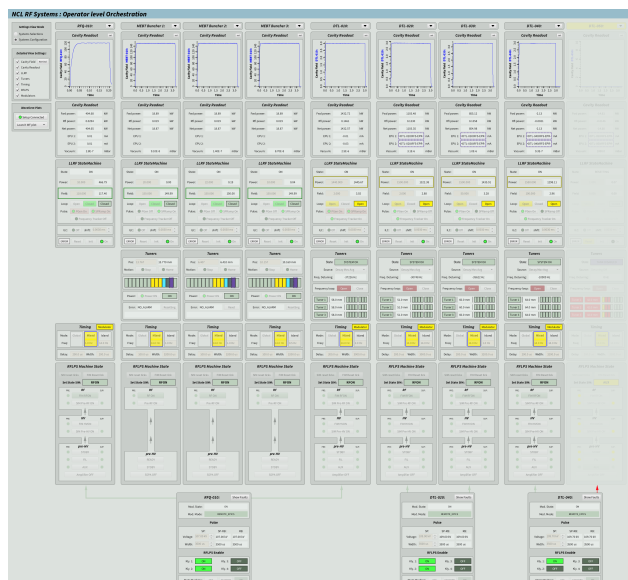

Figure 9: RF Orchestration OPI in a fully expanded view.

Figure 10: RF Orchestration OPI in an almost fully collapsed view.

As can be seen, the collapsed view offers the user a quick overview of the system states via colour codes based on the state relative to its expected state during normal operation:

- Red: System state is not OK (e.g. off or in fault).
- Yellow: System state might need attention, as it is not in the state it should be for normal operation.
- Green: System state is OK for normal operation.

This enables the user to quickly gain an understanding of the state of the systems as well as means to quickly interact with systems or launch OPIs with more advanced (less interacted with) settings.

CONCLUSION

A set of OPIs were created to give Operators a quick overview and simpler means to control a very complex machine that currently is in installation phase, meaning that the OPIs have to be quickly adaptable and dynamic to facilitate operational flexibility. For this purpose, the complex OPIs used for Navigating between other OPIs (Navigator OPI) and between different accelerator components (Accelerator Synoptic OPI) are successfully generated using Python scripts when a change is made to the accelerator lattice, a request to include more system species is made, or newly developed and deployed OPIs that Operators added are to be included. Therefore, the script-generated OPIs *Navigator OPI* and *Accelerator Synoptic OPI* are fully scalable to the full accelerator whilst the RF Orchestration OPI needs slight changes to some embedded system components in order to support e.g. 4-klystron-per-modulator systems and the Spokes RF systems.

Whilst the Python scripts for the Navigator OPI and Accelerator Synoptic OPI initially developed to be mainly used by a linear particle accelerator, the script is constructed such that it can be modified and extended to many other types of facilities.

ACKNOWLEDGEMENTS

The authors acknowledge the great support and collaboration across multiple divisions at ESS, including the Operations Division and the Integrated Control System Division, and to all colleagues reviewing and providing feedback both on the OPIs and on this paper itself.

REFERENCES

[1] K. Shroff *et al.*, “New JAVA Frameworks for Builing Next Generation EPICS Applications”, in *Proc. ICALEPCS’19*, New York, NY, USA, 2019, paper WESH1002, pp. 1497-1500. doi:10.18429/JACoW-ICALEPCS2019-WESH1002

[2] D. Nordt, “ESS Rules for the Visual Design of EPICS Operator Interfaces”, European Spallation Source, Lund, Sweden, Rep. ESS-4752055, 2023.