HOW TO IMPROVE INTERACTIONS WITH THE CONTROL SYSTEM

S. Deghaye, CERN, Geneva, Switzerland

Abstract

The availability of the LHC control system has been excellent in 2016, with only 0.11% of the machine unavailability attributed to Controls. Nevertheless, many other criteria can be maximised in order to improve the LHC control system and this paper focuses on the user experience. After having identified the main users, we detail some of the perceived problems, e.g. long development cycles, and areas where improvements can be made such as the standardisation of interfaces and integrated tooling. Finally, we propose organisational and technical solutions that we aim to apply in the near future in order to try to optimise the user experience.

INTRODUCTION

Several criteria can be used to evaluate the control system and the system availability is without question the highest-ranking criterion. Indeed, no matter how good your control system is in terms of, for example, features and performance, if it is not available to control your accelerator, its value is close to zero. 2016 has been a very good year for the accelerator control. According the Accelerator Fault Tracking (AFT), a mere 0.11% of the beam unavailability was attributed to the accelerator controls. Figure 2 is an extract of the system unavailability chart for the year 2016.

Other important criteria can be used when assessing the control system. For example, one could look at the financial aspects such as the hardware cost, taking into account both the cost of a new installation but also the cost generated by the periodic renovation one has to perform. One could also consider the software cost with license fees that one has to pay every year to use the commercial software used in the controls system. Looking beyond CERN, one could consider the exportability of the system to be an important aspect. Figure 1 depicts the author's subjective evaluation of the current accelerator control's infrastructure. This article focuses on yet another aspect: the user experience. Due to time and space constraints, we do not study specific applications, which are built using the infrastructure but instead look at the general user experience of the control system infrastructure as a whole.

USER EXPERIENCE

According to Wikipedia, the user experience (UX) refers to a person's emotions and attitudes using a particular product, system or service [1]. Other definitions highlight the human-product interaction, the quality of this interaction. In our context, it is worth noting that the interaction is not limited to human-computer. Even though usability is often seen as the most important aspect of UX, there are many more such as flexibility. Flexibility is understood to be how easy and quickly the system can be used and extended to accommodate new needs, with "new" being the keyword. Examples of new needs could be new ways to post-process data or new features to enhance the infrastructure or solve a common problem. For our work on the control system infrastructure, we look mainly at the usability and the flexibility. We concentrate on how the control system empowers users or impedes them in their daily job. Improving the user experience is a way to improve the control system, nevertheless, it must be done without degrading the other criteria, mainly the availability, from today's high-level.

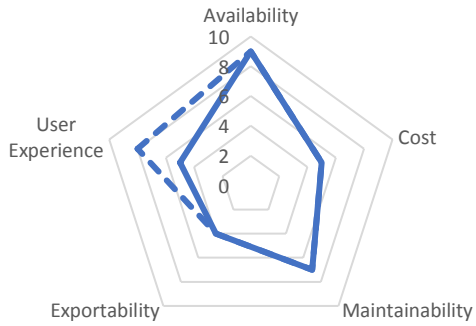

Figure 1: Subjective evaluation of the accelerator controls

Before going further, we need to define who are the control system's users. Accelerator operators and accelerator physicists (AKA MD users) are clear end-users of the controls solutions. But, as we talk about the infrastructure itself, we have also a lot of developers working at different levels of the infrastructure implementing specific controls solutions. The high-level application developers implement GUIs and scripts for interfacing with high-level services such as the logging, LSA, the post-mortem system, etc. Low-level software developers mainly work on FESA classes and kernel drivers with little to no interaction with the high-level services; at least until the operational deployment. Finally, the hardware developers design electronic boards with FPGA and implement their logic using HDL (High-level Description Language). Whenever the infrastructure or a specific solution does not work, operators call the support team for diagnostics and intervention (1st-line Diagnostics and controls experts). The examples above are by no means exhaustive and there are more user categories. When looking at the interactions between the control system and the users, it becomes clear that we should not define the users as community of human beings but rather take into account that a user has multiple roles that he/she will take depending on the task in hand. A person can be on shift for

the LHC, but alternating their roles and tasks between those of an LHC operator and those of a 1st-line diagnostic expert. The next day, the same person might be the machine expert post-processing data acquired during the shift. So, there are different interactions with the control system and different needs depending on the role.

PERCEIVED UX LIMITATIONS

By interviewing key user representatives, one can identify two common problems with the control system infrastructure. The first problem is its complexity and how it is exposed, and therefore the steep learning curve encountered when starting to work with it. The second problem is its lack of flexibility; which is both technical and organisational. In other words, the control system infrastructure is perceived by the users as heavy and complex. In addition, the inadequacy of some tools for common tasks is often mentioned. Tools have to be taken in a general sense as this comment applies to GUI tools such as, for example, the APEX CCS editors but also the development languages available.

The current situation is not surprising. Indeed, the recent years were focused on availability and maintainability of the infrastructure with renovation and consolidation projects such as ACCOR & InCA. Therefore, even if there are still about 900 FECs to be renovated before end of LS2, we have a system that is appropriate for long-lasting, stable operation phases. In addition, the release and deployment phases are also well-organised and the post-technical-stop recovery time is shorter than a few year ago. Furthermore, as the first years of LHC operation were very important, the focus was more on the accelerator operator role than on the other roles such as the software developer. The consequences of those choices are that the control system is less well-adapted for a quick-and-dirty test, fast-iteration development, or simply final validation using the whole infrastructure.

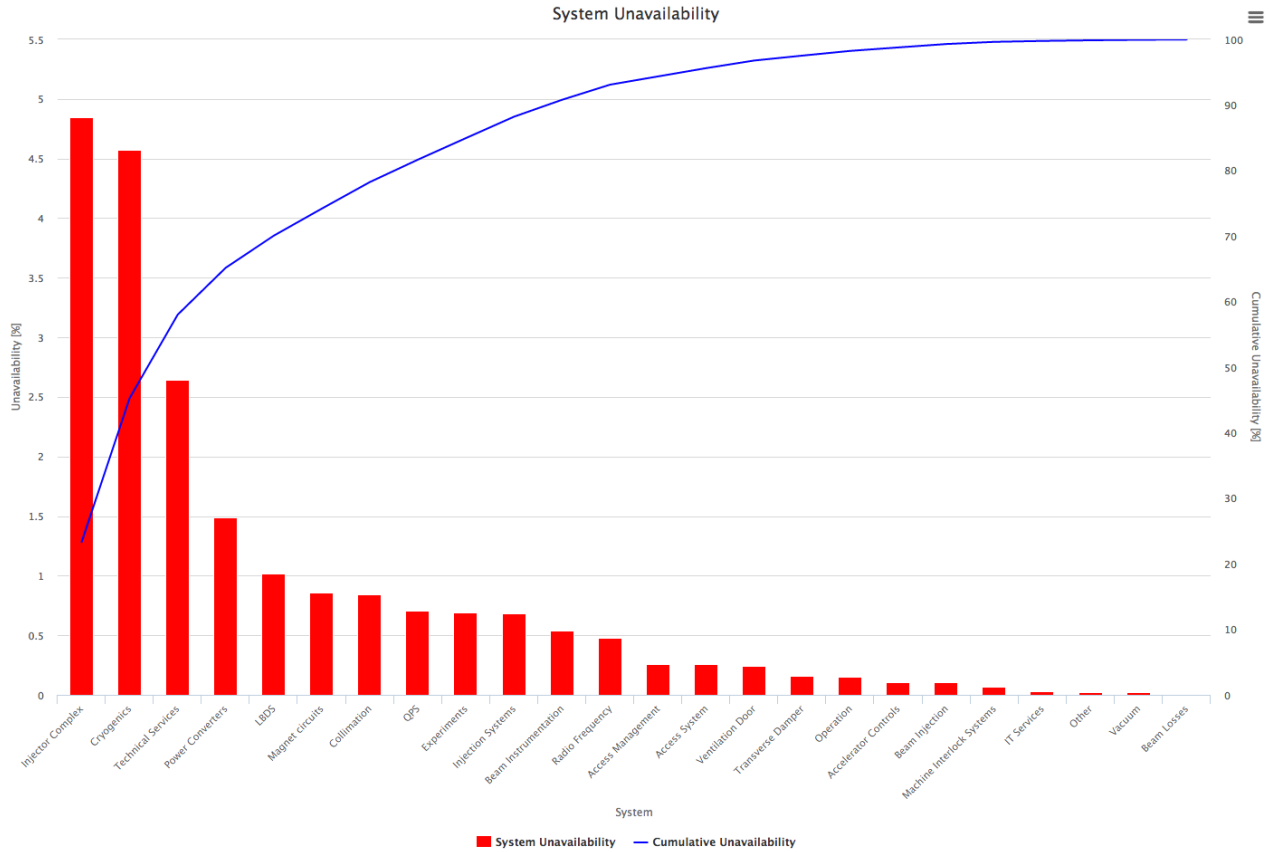

Figure 2: Statistics of the LHC faults in 2016. Extracted on the XX-XX-2016 from AFT

The root causes of the problems mentioned above are well known. The control system's complexity is too exposed to the users. It lacks homogenous interfaces and the integration of the different layers could be improved. The tools, end-user applications, and programming languages are inadequate for specific tasks typically performed by some expert roles.

Today's control system infrastructure can be seen as a set of independent components. At runtime, those components collaborate to achieve the expected behaviour but from a developer or accelerator expert's point of view, the whole internal structure is exposed and a (very) good knowledge of the different components and layers is required. Furthermore, the integration could be improved as today's level of integration has two main consequences. Firstly, simple tasks can require many actions and there are no straightforward, easy-to-remember connections between them. The workflows are sets of steps without

clear relationships and are based on tools as different as web applications and shell scripts can be. Secondly, the lack of integration leads to disparate and sometimes incompatible feature sets between the control layers. For example, some extensions were implemented at the front-end layer for specific use-cases but these extensions are incompatible with the high-level concepts. For a seasoned developer, this will not be a problem but a junior programmer can potentially lose a lot of time when he realises that incompatible features are used and his design needs to be reworked.

With respect to the interfaces' homogeneity, there is no common API to access all of the high-level services, even for the basic use cases. To give an example, one cannot subscribe to a power converter in the SPS for all the SFT cycles since yesterday with a single consistent API. Instead, it is necessary to fetch the past data from the logging service and then subscribe to the actual device for live data using another API.

All the points mentioned above are neither critical nor blocking but as the system's operational maturity is reached, improvements in those areas have started or are in the pipeline. In the next sections, we will go through different ways of improving the usability and the flexibility of our system. Some are still at the level of suggestions but they will clearly shape the evolution of the control system over the coming years. Other proposals are already being taken into account as part of recent developments.

IMPROVING USABILITY

In order to improve the usability of the controls tools and of the system in general, more emphasis has to be put the on the control system's use-cases rather than focusing on a specific service's needs. For a developer, that means a better full-vertical integration with compatible features that make sense throughout the layers of the control system. Another example are the configuration tools. They should not be a set of heterogeneous service-specific tools but instead a single tool, or at least tools with a single-entry point, which guides the user through the process required to assemble all necessary information. Furthermore, if different tools are required to perform a task, these tools should be linked so that the end-user does not need to remember the list of tools to be used.

This is the approach taken in the new Controls Configuration Data Editor (CCDE). Figure 3 depicts a mock-up of the new hardware configuration editor, which is part of the CCDE. The top panel of the interface assembles information from different sources so that the end user does not need to gather data from different tools, web pages, etc.

Whenever a new service is put in place, it will be integrated into the existing tool rather than providing a new service-specific tool. This better integration immediately helps to reduce the so-called configuration marathon. It is important to note that this approach does not mean that the resulting tool is a bulky application difficult to maintain. Indeed, the modern web-based applications, distinct tools can be seamlessly integrated.

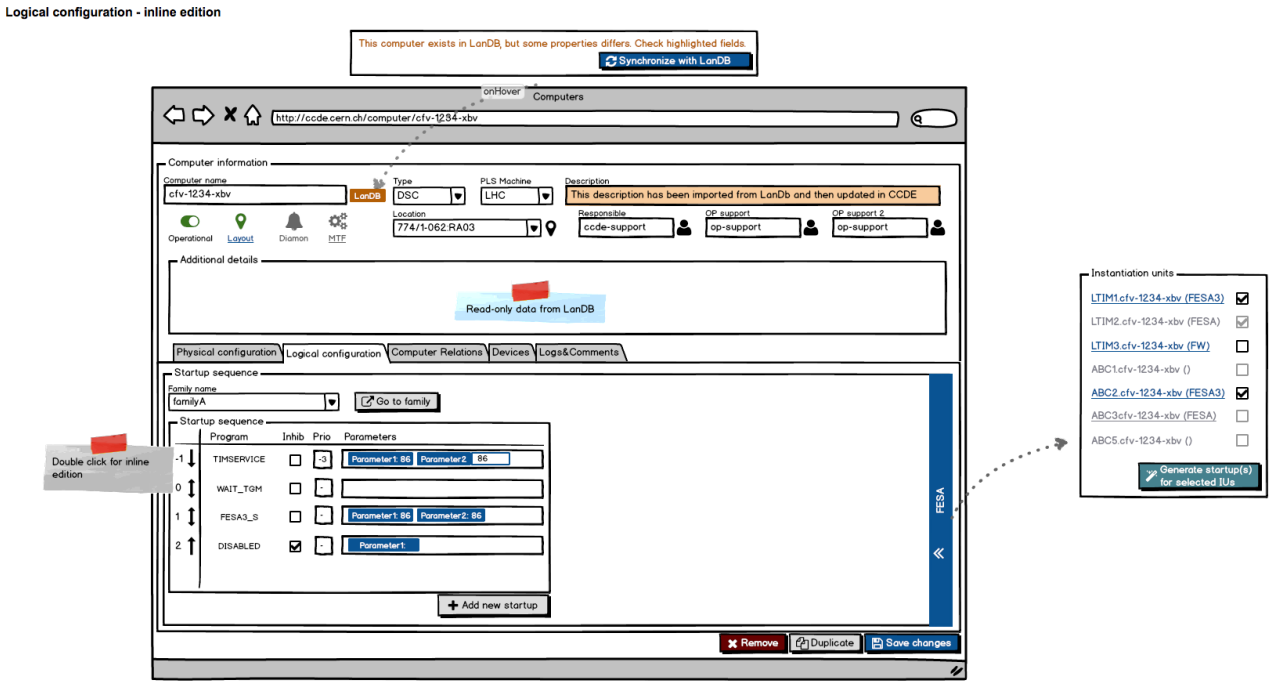

Figure 3: Mock-up of the new Controls Configuration Data Editor (CCDE)

In addition, we want to apply a principle known as "convention over configuration" to the control system's configuration. Today, every single bit of the control system can be configured and most of it with opt-in behaviour (i.e.

you need to indicate that you want to use a given service). In recent years, more general attributes, such as the control device state, have been introduced. Thanks to this simple piece of information, more conventions will be put in place and more default behaviours can be inferred from it. For example, it is planned to rely on the device's state attribute to decide by default whether the alarms should be monitored, which settings to be managed, etc.

Returning to the interfaces' homogeneity, we have to emphasis the difference between the needs of the end-users and the need for a proper maintainability; which can be seen as opposing forces. The fact that the control system is a set of islands is actually a benefit from the maintenance point-of-view. Every complex system should be easily decomposed into simpler modules in order to manage the complexity. That being said, what is clearly missing in the current system is a common interface to access the different services (islands) as, from a user perspective, the source of data (FESA, InCA acquisitions, LSA historical settings, CALS, Post-Mortem, etc.) should not matter. The control system is built around a data model that commonly referred to as the device/property model. In a few words, the device-property model says that the independent entities are devices and the devices have properties that can be read and written. Obviously, this model is not going to cover the most exotic cases but this is the direction we want to take for the future, especially whenever we have opportunities to renovate services such as the CERN Accelerator Logging System (CALS). We will know that we have reached our goal when it will be possible to seamlessly retrieve logged data from two days ago and live data from the accelerator with the same API.

To finish the discussion on usability improvement, we need to look at it from a developer's perspective and how easy it is for them to quickly iterate in their development cycle, as well as validate their development without impacting the operational accelerators. The main issue is that the system has been solidified for many years to ensure excellent operational reliability but shortcuts required for agile development are not in place yet. Furthermore, and rightly so, developers are encouraged to work on the General Purpose Network (GPN) where not all of the services are available. In recent years, more and more effort has been invested in the setup and use of testbeds. In BE-CO, most of the low-level frameworks and libraries are extensively tested on the so-called Controls TestBed (CTB). With an even bigger ambition, TE-MPE is putting in place a complete hardware and software test bench as described in [2]. There is also an ongoing effort to provide high-level services on the non-operational GPN network so that validation can be done without having to pass through formal steps such as code release and deployment. Finally, the topic of simulation has recently received more attention and many usability issues could be solved by providing out-of-the-box simulation modes whereby one could limit its dependencies on external systems. For example, a simulation mode for the timing system would simplify the testing with different beam sequences without depending on the actual accelerator schedule.

IMPROVING FLEXIBILITY

Several initiatives have been launched in the recent months to improve the flexibility of some of the controls components. Furthermore, we would like to experiment with different organisation of the work that, among other things improve the organisational flexibility. One of the areas that needs improvement is the rapid application development, i.e. solutions that provide an easy-to-use language for the situation where a quick test or validation must be done and where a full development is not practical. The Python language has been seen by many as a valid solution for cases such as Machine Development (MD) slots and also for low-level hardware validation. Inspector, a tool to quickly design GUIs, is another successful example on how to add flexibility in the controls offering.

In both cases, we want to work differently and build stronger collaborations. In the past, one of the central controls groups would have taken over the support of the technology or the tool. Indeed, we believe that the end-result can be much better if all CERN users could, if they wanted and had time to, contribute to the tools. Of course, any attempt to modify the way we work and collaborate cannot be done without ensuring that the current levels of availability are kept and bearing in mind our long-term needs in terms of maintainability. This change of organisation means that we depart from the usual client/provider approach. This approach does not scale very well and introduces delays between new needs and the availability of the solution, in other words, organisational inflexibility. This problem has been solved for a long time by open-source communities for software as big as Linux. Nevertheless, one must keep in mind that we have specific constraints in terms of long-term stability and therefore responsibilities must remain clear. Figure 4 depicts the workflow of two Tango products Taurus and Sardana. This workflow relies on modern software engineering tools and concepts such as Git, pull requests (PR), and code reviews. We believe that such an approach should be attempted for the support of the Python technology. Instead of expecting a central entity to develop and support everything related to that technology, we build a community around a focus group where people can share their experience, problems and solutions but also decide collectively on the creation of new components or libraries and the evolution of existing ones. The support is then given by the main authors but the community acts as back-up should the experts be unavailable (holidays, sick leave, etc.). To ensure that the required discussions take place and that the community is kept alive, a central controls group such as BE-CO should organise the focus group. If the approach is successful, the same should be applied to other core frameworks such as FESA. One could very well imagine that new features are introduced by a framework user after discussion and validation by the core team. Later, the developer submits (aka make a pull request in Git terms) to the team that performs a code review and verifies that the long-term requirements (code quality, tests, etc.) are satisfied. As the last step, the new feature is available in

the latest release and the whole community can profit from it. Compare to an approach where all the requests are sent to a single team, prioritised and worked on by them, it is evident that a more collaborative approach brings flexibility in our organisation.

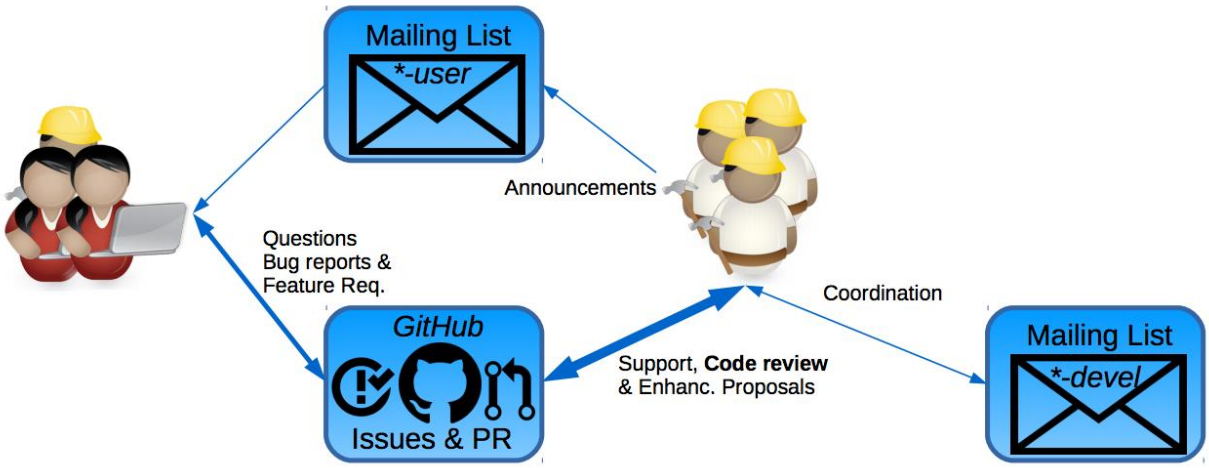

Figure 4: Taurus & Sardana community development (Courtesy C. Pascual-Izarra et al.)

ACKNOWLEDGMENT

The author would like to acknowledge all colleagues who contributed their inputs, examples and reflections.

REFERENCES

[1] User Experience definition, Wikipedia, https://en.wikipedia.org/wiki/User_experience

[2] M. Zerlauth, Project Roadmap for MPE Testbed in b272, EDMS Document no 1740427.