CONTROLS - AN OP PERSPECTIVE

K. Fuchsberger

Abstract

Based on controls expectations from previous years, this talk will summarize which of these issues were fulfilled during the last years and which are still pending. Actual problems and potential improvements of the actual LHC control system will be discussed with the goal of outlining a coherent longterm vision.

INTRODUCTION

During the past years, the LHC control system has evolved to a very good state. In general, the control system is very stable and all operational scenarios are well manageable, even including some more 'uncommon' scenarios such as 'Ramp & Squeeze'. Additionally, a lot of improvements and cleanups of APIs were completed during Long Shutdown 1 (LS1).

Having in mind the very good quality of service, the question might be asked, if changes are required at all? This question must be clearly answered with 'yes', with the following arguments:

- Improvements have to be made to avoid mistakes and increase availability [1].
- Changes are necessary to (carefully) evolve and update the control system; The world around changes (e.g. new computer language versions, new processors, etc.).

Despite the fact that changes are required, sometimes it happened in the past that changes were introduced which were not really requested by operation but nevertheless had a big impact on many layers. Examples are:

- Breaking API changes in general
- LSA refactoring (with breaking changes)
- Logging Service API (changed twice)
- Changes in various FESA classes
- incompatible rewrites of middleware frameworks (FESA and RDA).

Although it is also understandable that such changes might be necessary from time to time, the main aspect to improve here is communication and transparency which is required in advance to reduce the unforeseen impact of such changes.

REQUESTS FROM PREVIOUS EVIAN TALKS

We mentioned in the previous section that some changes were implemented but not requested. Therefore, it is worth looking at requests from OP and their completion rate. Since there are no official statistics available for this data, we will take a very simplistic approach and look at the requests which were mentioned in talks from previous Evian workshops [2, 3]. Figure 1 shows the summary of this analysis, giving the percentage of implemented and not-implemented features, requested during the years 2010 and 2011. A more detailed list is shown in Fig. 2.

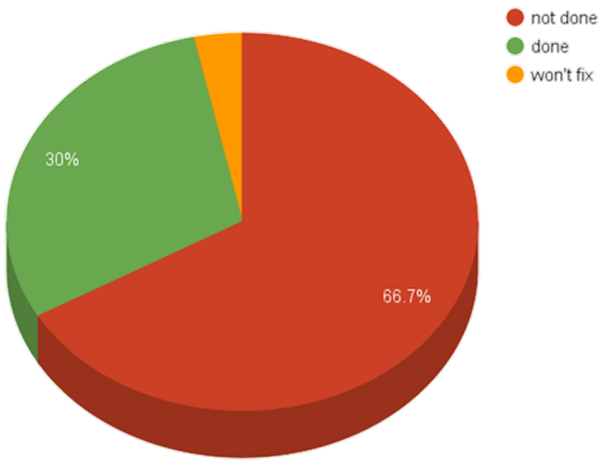
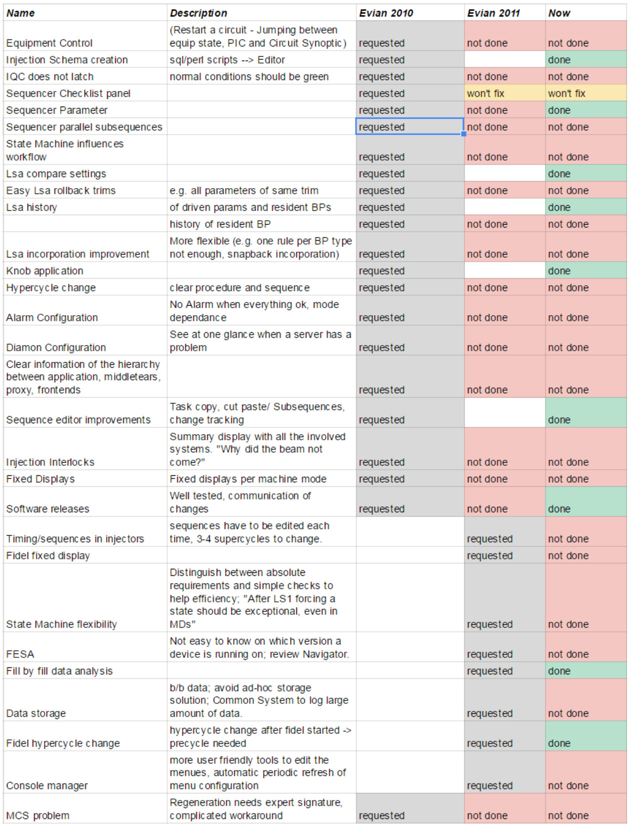


Figure 1: Features mentioned in Evian talks related to controls in the years 2010 and 2011 and the percentages of those which were implemented.

It should be noted that this is a very rough analysis and the reasons why certain features were implemented and others not were not analyzed. The only deduction is that about a third of the features requested in previous Evian talks were implemented.

WHAT OP WANTS - THE LHC TOP 5

Having looked at the requests from previous years, it is the next step to see what features the LHC operations team would like to have for the future in order to operate the machine in a more efficient way. To answer this question, a series of meetings were conducted with the goal to find a consensus and define the next goals for the LHC control system for the near future.



Name	Description	Evian 2010	Evian 2011	Now
Equipment Control	(Restart a circuit - Jumping between equip state, PIC and Circuit Synoptic)	requested	not done	not done
Injection Schema creation	sql/perl scripts --> Editor	requested		done
IQC does not latch	normal conditions should be green	requested	not done	not done
Sequencer Checklist panel		requested	won't fix	won't fix
Sequencer Parameter		requested	not done	done
Sequencer parallel subsequences		requested	not done	not done
State Machine influences workflow		requested	not done	not done
Lsa compare settings		requested		done
Easy Lsa rollback trims	e.g. all parameters of same trim	requested	not done	not done
Lsa history	of driven params and resident BPs	requested		done
	history of resident BP	requested	not done	not done
Lsa incorporation improvement	More flexible (e.g. one rule per BP type not enough, snapback incorporation)	requested	not done	not done
Knob application		requested		done
Hypercycle change	clear procedure and sequence	requested	not done	not done
Alarm Configuration	No Alarm when everything ok, mode dependance	requested	not done	not done
Diamon Configuration	See at one glance when a server has a problem	requested	not done	not done
Clear information of the hierarchy between application, middletears, proxy, frontends		requested	not done	not done
Sequence editor improvements	Task copy, cut paste/ Subsequences, change tracking	requested		done
Injection Interlocks	Summary display with all the involved systems. "Why did the beam not come?"	requested	not done	not done
Fixed Displays	Fixed displays per machine mode	requested	not done	not done
Software releases	Well tested, communication of changes	requested	not done	done
Timing/sequences in injectors	sequences have to be edited each time, 3-4 supercycles to change.		requested	not done
Fidel fixed display			requested	not done
State Machine flexibility	Distinguish between absolute requirements and simple checks to help efficiency; "After LS1 forcing a state should be exceptional, even in MDs"		requested	not done
FESA	Not easy to know on which version a device is running on; review Navigator.		requested	not done
Fill by fill data analysis			requested	done
Data storage	b/b data; avoid ad-hoc storage solution; Common System to log large amount of data.		requested	not done
Fidel hypercycle change	hypercycle change after fidel started -> precycle needed		requested	done
Console manager	more user friendly tools to edit the menues, automatic periodic refresh of menu configuration		requested	not done
MCS problem	Regeneration needs expert signature, complicated workaround	requested	not done	not done

Figure 2: Detailed list of controls related features mentioned in Evian talks in the years 2010 and 2011.

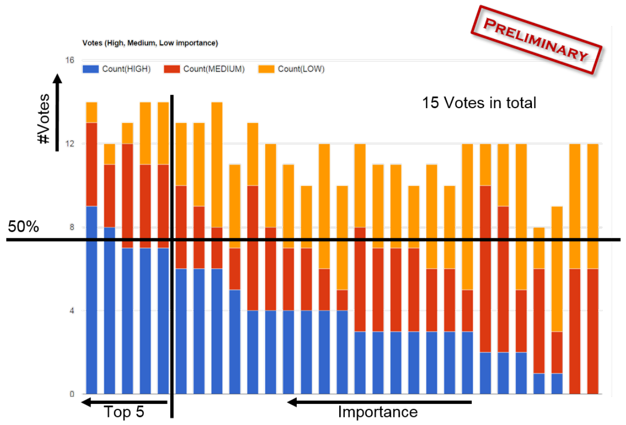


Figure 3: Distribution of the voting result, ordered descending by number of 'HIGH', 'MEDIUM' and 'LOW' importance.

Note: The results given in this section, are those which were available at the time of the workshop. In the meantime, the priorities and results have evolved. Still, we will only present the state at the time of the workshop.

In preparation for these meetings, the following three questions were asked to the the operations team:

What do you consider as the . . .

- . . . most important features/properties/components that an ideal control system should have? (Long term)
- . . . most important improvements that should be made to the available tools?
- . . . most important issues that should be fixed as soon as possible?

Since time was short before the workshop, and no consensus could be reached during the meetings, a simple email vote was conducted to get an impression on the priorities. The result of this vote forms the basis of the following top 5 priorities at the time of the workshop. The distribution of the vote result is shown in Fig. 3.

It is clear that the distribution is quite flat and that there is no real significant 'winner' in this vote. This distribution clearly points to the fact that the operation crew is far from a consensus in this matter and some further discussion has to be completed before a clearer statement on the OP priorities can be given. (NOTE: Further meetings were made in January, with different outcomes - not discussed here).

For the following we took a rough cut around the point where about 50% of the voters considered these improvements as 'VERY IMPORTANT'. All the features above this threshold are presented in the following in more detail.

1 - Improved Automation/Sequencer/Scripting

The item which was voted as priority number 1 was very broad. This is most probably also the reason why it received so many votes. So the result indeed is quite doubtful and of limited use. For a follow-up, this clearly has to be split. Still, we describe here the main proposals for improvements which were mentioned within this context:

Sequencer Execution:

- (Automation) Parallelism would be useful.
- Better overview would be good when several sequences are running.
- Several small details (e.g. Quirky Windows behaviour, Better error presentation).

Sequence Editing:

- Long time/chain from idea to operation (every change in a task needs a sequencer release).
- Tools that make it easier to refactor between different layers (Sequence ↔ Task) would be useful.
- Scripting (More flexible - and more dynamic - way to create scripts/macros/sequences). E.g. for Easy way to formulate commissioning tests

Settings vs. Sequences:

- Clear separation between settings and sequence would be preferable, instead of hardcoded values in sequence (devices, contexts, values).

- New concept of Templates/Operational Scenarios (certain modes of operation (e.g. Proton Physics, Lossmaps, VdM scans)
- Avoid e.g. different sequences for different particle types. (New copies with slight changes) Clearly see the discrepancies when something is wrong.

2 - QPS/PIC/Equip State

This item covers the whole problem of dealing with circuits and the circuit protection system from an operational point of view. Features/Improvements which were mentioned in this context are:

- As soon as the PIC permit is lost, the root cause could be displayed immediately. The Post Mortem system knows it, but it takes long. Could this be faster?
- Replacement for QPS macros? e.g. Sequences or new mechanism?
- Better integration of WinCC GUIs in other control systems.
- Simple overview would be good; Dig down in case of problems.
- Easy tool for Circuit (+QPS) resets would be appreciated.

3 - Improved Filling Diagnostics

In the current way of operating the LHC, a lot of time is lost during the injection phase [4]. Especially, when the beam is requested, but is not injected, the problems are hard to diagnose.

One way to improve this situation would be to introduce another display which could show (potential) problems which led to the situation that the last injection did not happen. A later version could most probably even predict if a following injection would work.

4 - Easily usable tool to move collimators

The current available tools to move collimators are not easily usable and also supposed to be only used by experts from the collimation team. However in several situations (e.g. Commissioning phase, Loss maps) situations occur where collimators have to be manipulated by the operations crews. Because of the expert nature of the applications and the rareness of these situations, performing this task is difficult and potentially dangerous.

Also e.g. moving collimators to the parking position is often needed but requires complicated acrobatics of making some Beam Process resident plus pressing the right combinations of buttons in EquipState. A small operational application with a subset of the expert functionality and eg. a set of predefined scenarios (Move to parking, Symmetrize TCTs) would help to improve this situation.

Another useful tool would be a small display where to see the collimators (probably w.r.t. the beam). Such functionality was implemented once in the aperture meter, but was never made operational.

5 - Improve Window Management on Consoles

This topic is on the table since 2010 [2]. This point is all about operational efficiency on CCC consoles. Remarks which were brought up in the survey are:

- Rationalization of the space on the consoles, avoiding overlapping windows. Probably a different window manager could help here.
- Perspectives per activity: e.g. Injection, Ramp, Lossmaps, Powering. The beam mode (or later the state machine) could decide what gets displayed and where.
- Autostart of a predefined set of applications on consoles (e.g. after a reboot).
- Keyboard Shortcuts (e.g. new logbook entry, search for an application).

DEVELOPERS VIEW

In addition to the operational perspective, it is also very important to take the view of a software developer. This is critical, because a big percentage of the applications are developed and maintained within the operation team itself, by operators and EICs. Figure 4 shows the percentage of origins of the most important GUIs, visible on the LHC consoles. Even if this only is a very rough number because of course the underlaying software (servers, FESA classes) is most of the time from other sections, it at least emphasizes the relevance of software developed in the OP group.

In the following we will discuss the status and potential improvements as seen by this developers perspective.

Layers

Currently, in our control system environment, developing a high level controls application is not straight forward. In order to achieve different tasks, different underlying components have to be accessed. For example:

- **Settings:** In order to store settings for devices, change them and drive them to the actual hardware, the application has to access different services of LSA.
- **Measurements:** To get (or subscribe to) measurement data, the application has to access devices (mostly FESA) directly and know the structure of the underlaying control system.
- **Historical Data:** To get historical data, an application has to access the accelerator logging service (CALS) and has to know how the logging variables are related to devices and properties.

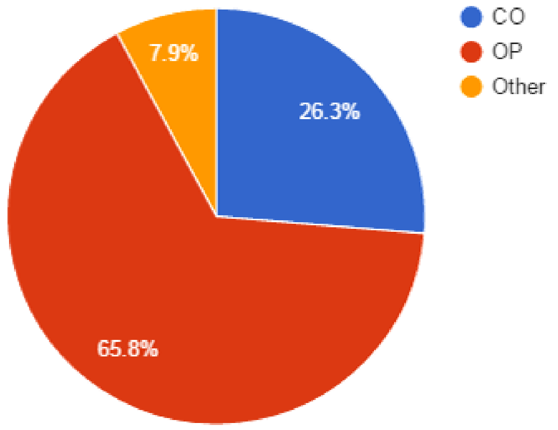


Figure 4: Distribution of origin of the applications mostly open on the CCC consoles. The three parts are number of applications written by members of the operations crew (OP), members of the controls group (CO) and others (Other).

Through all the mentioned channels, a software developer receives different data types, data of different abstractions and conversions are necessary all the time, which are reimplemented in every application. This encourages copy-paste code which is error prone and hard to maintain.

An additional problem is that different parts of the control system are designed by different people and sections within the controls group without a common vision. So the proposed solution for a given problem strongly depends, in practice, on which section/person is asked first and therefore several times functioniality is not implemented in the most efficient manner.

The first thing which could improve this situation is to put in place a better layering. While such layers are called ’business layers’ in industry, we will refer to it as ’physics layer’ in the following. This concept is roughly sketched in Fig. 5.

Such a physics layer would ideally have the following properties:

- Domain Driven Design
- It eases testing, by facilitating the replacement of certain layers by testing layers.
- It makes writing applications (clients) damn easy.

The last point is illustrated in Listing 1. It shows a few lines of (fictionally) getting the tune of one beam of the LHC or subscribing to it.

Listing 1: RefOrbitService usage example

```
Tune tune = get(HORIZONTAL, TUNE)
        .of(LHC, BEAM1);
on(HORIZONTAL, TUNE).of(LHC, BEAM1)
        .subscribe(System.out::println);
```

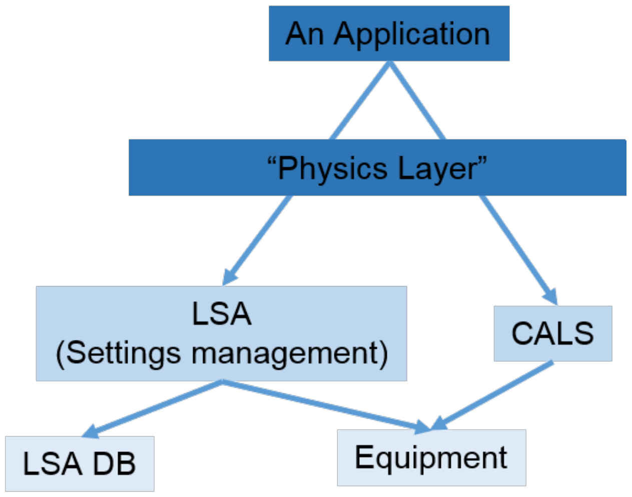


Figure 5: Sketch of a layered control system, with a ’physics layer’ in place (which is not the case at the time of writing).

THE GLOBAL VIEW

As mentioned already in one of the previous sections, one of the current problems in our control system is a lack of global vision. Different parts of the control system (GUIs, servers, devices) are developed in ’islands’ (small teams or - most of the time - individuals). This introduces incoherence and code duplication.

To counteract this tendency, it is necessary to develop a global long term vision. This could be based (amongst others) on basic principles listed below. For example, the control system should

- be intuitive,
- be consistent,
- be simple on top, easy to dig down,
- be built to be tested,
- be built to grow,
- have the right dose of automation.

FEARLESS TESTING

Most of the parts of the current LHC control system are hard to test without beam. Even worse, sometime it is even unclear if testing a certain component would influence the production environment (e.g. could potentially dump the beam) or if it is safe to test. This leads to a hesitation of people to test their code. This slows down development, prevents change and fast deployment and thus lets the codebase degrade.

To escape this problem a lot of effort has to be put in the future into untangling components, providing consequent layering and making it simple to develop simulators and testbeds for control system components. The goal should be a testing environment which allows each developer to test their components without the fear of interfering with operations.

LHC OP SOFTWARE STATUS

As mentioned in a previous section, a lot of software is developed within the OP group itself. Figure 6 illustrates the actual status of the sourcecode of software for the LHC which is maintained by the OP group (in this example, people from the LHC and SPS sections). The plot was preduced using metrics from sonarqube [5] and shows the number of lines of code (LoC) and the estimated technical dept in mandays.

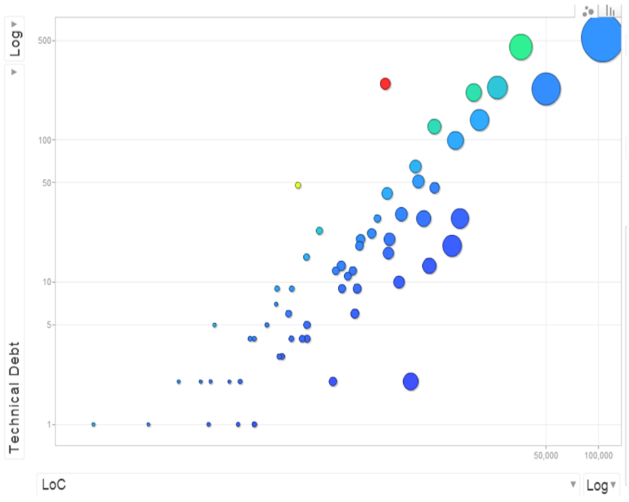


Figure 6: Technical debt of projects vs Lines of code (LoC).

Actual Status In numbers, the situation is as follows:

- 500.000 lines of code (800.000 lines in total)
- 13.6 man-years of technical debt
- 6(LHC) + 2(SPS) people writing software
- → 8.4 minutes of tech dept per line of code!

In addition to the mentioned existing debt, there are several projects which are under discussion to be taken over by OP, because there is no other possibility to maintain them. The situation of these projects are illustrated in Fig. 7. In numbers this reads as follows:

- 100.000 Lines of code, 160.000 lines in total (20% of actual codebase)
- 2.5 man-years of tech debt

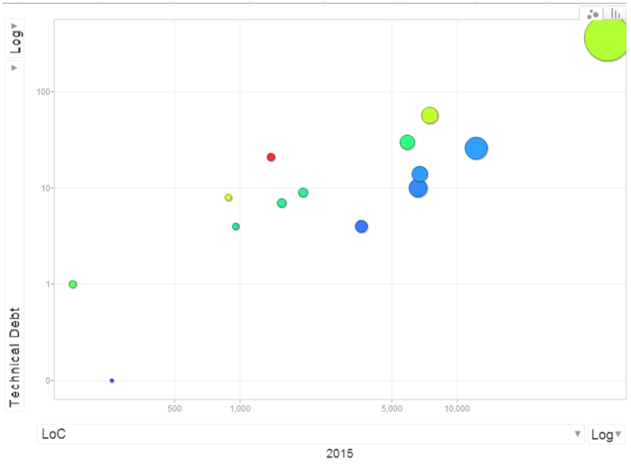


Figure 7: Technical debt vs lines of code (LoC) which potentially should be maintained by the OP group in addition to the actually maintained projects.

- 2 students projects + 1 external SW stack.
- → 8 min/LoC

It is interesting to note that the incoming projects have a comparable quality than the currently maintained codebase (roughly 8 min technical debt per LoC).

TACKLING THE PROBLEM

One of the reasons for the bad state of the codebase is the lack of continuity within the projects. Part of this comes from the already mentioned lack of global vision, while another part can (without proof) be attributed to the current 'process' of software development in OP: Very often it happens that people (with a lack of software experience themselves) get in students which are told to write software. Even if these students do their best, the outcome is very often avarage or below, due to a lack of close supervision.

The currently proposed approach to tackle these problems is to work more in teams. Working in teams improves knowledge sharing and allows people to learn from each other. For sure there is also the cost of communication overhead. However, in the current situation, the advantages seem to outrule the costs.

When forming teams, it is difficult to find a way to work together on one project, because each of the (future) team members is attached to 'their own' project and will see this as most important. The priniciple idea here is to see all the projects coming from the individual people as kind of 'one project' and work on this single project together, driven by priorities. This approach is sketched in Fig. 8.

The same approach was already successfully implemented in the MPE-MS section and also proved to be very useful when a new testing system for the LHC orbit feedback system was developed earlier in 2015. The advantages

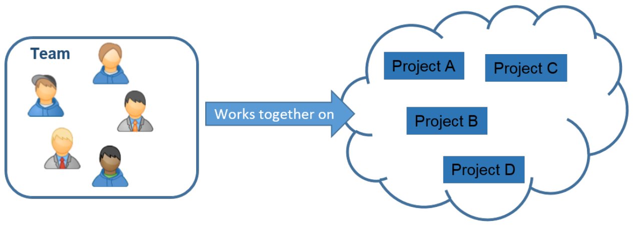


Figure 8: Several individuals form a team and work together on all their projects.

are:

- it fosters knowledge exchange,
- better software is produced,
- it allows to focus which improves productivity by reducing context switching
- it is more fun for the people involved.

However, part of this is very hard to do within the OP group, because of the shiftwork. It is especially difficult to ensure …

- …continuous progress,
- …continuous supervision,
- …continuous support.

The only way to consolidate this approach is to encourage collaboration with other sections. In this spirit, several development iterations are in preparation together with the TE-MPE section for 2016. Also the CO-APS comitted to give a try to this approach later in 2016.

SUMMARY, FOLLOW UP AND OUTLOOK

The LHC control system is in a good shape. Everything is working well and all (even complicated) physics cases are well covered. This is the time of careful evolution. In this paper, we presented two major viewpoints:

As Control System Users, we would like a Control system that behaves as we expect it to, prevents us from making mistakes and evolves according to our priorities. At the same time we know that we have to improve in defining our priorities.

As LHC-OP software developers, we would like to develop in teams together with developers of the controls group, have intuitive domain specific layers to program against and be closely involved in evolving the control system in design and implementation as well as strategic decisions. At the same time we are conscious that we have to improve our software-development skills.

After the actual Evian workshop, several additional meetings were conducted and our priorities were refined. Several collaborative sprints were completed together with the TE-MPE section and another effort which involves CO developers is in preparation at the time of writing.

For the actual year, the aim is to start earlier with discussions and prioritization to be ready for the winter shutdown 2016/2017.

REFERENCES

[1] A. Apollonio, “2015 availability analysis”, these proceedings.

[2] D. Jacquet, “Software and Controls Issues”, Proceedings of Evian Workshop 2010.

[3] D. Jacquet, “What we Want”, Proceedings of Evian Workshop 2011.

[4] D. Jacquet, “Injection”, these proceedings.

[5] http://www.sonarqube.org