SOFTWARE TOOLS FOR MPS

Kajetan Fuchsberger (CERN, Geneva, Switzerland)

Abstract

While dedicated Hardware systems protect the LHC against different types of failures, the role of software systems in the environment of Machine Protection is more in the area of configuration, supervision and diagnostics. This paper will present ideas for improvements on some of those systems as well as visions for further developments. A dedicated focus will be given to tools that shall improve the reliability of Machine Protection Systems (MPS) commissioning steps and other software improvements which could prevent human errors during operation, and thus increase availability.

MOTIVATION

For the commissioning of the LHC magnet circuits (in view of the large amount of work, i.e. about 7,000 individual tests), a lot of effort was put in the automation of tests, starting right from the beginning of hardware commissioning in 2005 [1]. Although slightly different (less tests, more manual tests), the commissioning of machine protection systems involves similar steps: Planning an appropriate sequence of tests, executing functionalities of the hardware and verifyinge the results (mostly manually). This similarity is the main motivation for the following proposals, as it seems appropriate to re-use the developed tools for commissioning of machine protection systems. This will be covered in the first part of this paper.

The second part of the paper describes ideas that should help to detect problems earlier during normal operational periods. The last part gives a short reminder on the Aperturemeter, which turned out to be a very useful tool during MPS commissioning and whose future is somehow uncertain and thus is worth some dedicated attention.

TESTS AND PROCEDURES

Current status

In the previous years, the progress of the MPS commissioning was tracked by the usage of a simple sharepoint site. Despite the simplicity of the usage, this solution had several disadvantages. Amongst them:

- The order of the tests could not be enforced at all. The scheduling was more or less done 'on the fly' by people on shift, based on their personal best knowledge.
- Nothing enforced that the tests were done at all.
- It was not possible to get a real overview of what was done already and what still had to be done.

Figure 1 shows an example view of the mentioned sharepoint site.

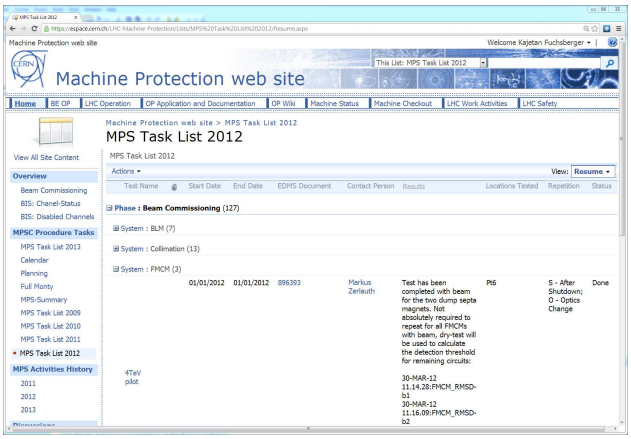

Figure 1: A screenshot of the Sharepoint site used in 2012 to track the MPS commissioning status.

The AccTesting Framework

The AccTesting framework ('AccTesting' in the following) was originally designed with the execution and tracking of tests for LHC hardware commissioning in mind. Nevertheless, since it soon turned out that a more general approach was appropriate, the goal was soon changed to create a general framework for the execution and tracking of tests for any kind of accelerator systems. In the following we will focus only on the explanation of those aspects that are necessary to understand the application of this framework for the use in commissioning of LHC machine protection systems. A more detailed explanation can be found at [2].

The framework is able to deal with a high workload and enables its users to work in parallel. Furthermore, it prevents execution conflicts and provides the current test status information to all of its users. A general overview over the architecture of the framework is shown in Fig. 2. The central point is the AccTesting server. The test execution and analysis results are stored in a database that only the server may access. The server itself is not aware of any specifics of the tests it handles. The test execution servers and the result analysis components are connected to the server with a plug-in like system. Each of them can handle a specific type of tests. If the main server wants to start the execution or analysis of a test, it provides each of the plugged-in test handlers with the test information, which in turn decide if

they are able to handle the test. Once a test handler has accepted a test and started the execution or analysis, the main server will regularly poll it to retrieve the test status and result.

The AccTesting server can be accessed simultaneously by several users through the use of a specific Graphical User Interface (GUI). The AccTesting GUI displays all the information about the currently executing tests and scheduled tests. In this sense it replaces the former test tracking web pages. Furthermore, it allows to enqueue a scheduling request to the AccTesting server directly from within the test plan view. A sample screenshot of the GUI is shown in Fig. 3.

The AccTesting server is designed in a very robust manner. It can deal with unexpected behavior from its plugged-in test handlers, errors in the control GUI, incomplete test results and many other issues like a sudden crash of the virtual machine. Furthermore, it provides a robust scheduler which is responsible for executing the enqueued tests in the most efficient way, while respecting the correct order together with all the constraints and preconditions. Another interesting feature of the framework, which makes it an interesting candidate for the tracking of MPS commissioning, is the integrated statistics functionality. This makes it very easy to get an overview of the actual progress of a commissioning campaign. A screenshot of the statistics view is shown in Fig. 4

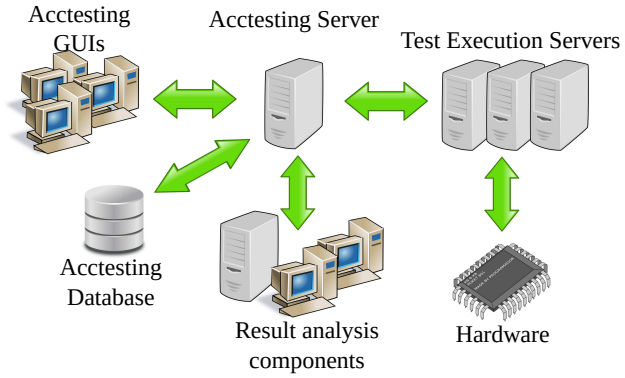

Figure 2: Components of the AccTesting framework.

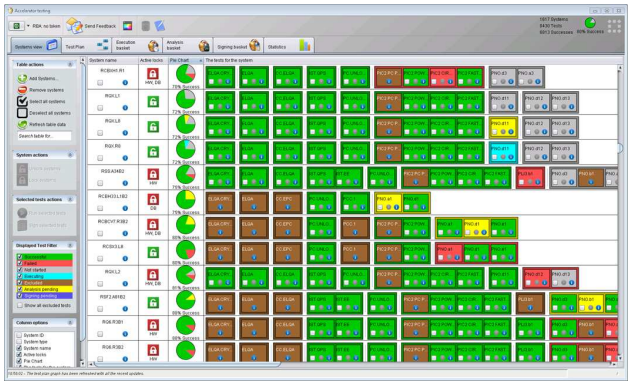

Figure 3: A screenshot of the graphical user interface (GUI) for the AccTesting framework.

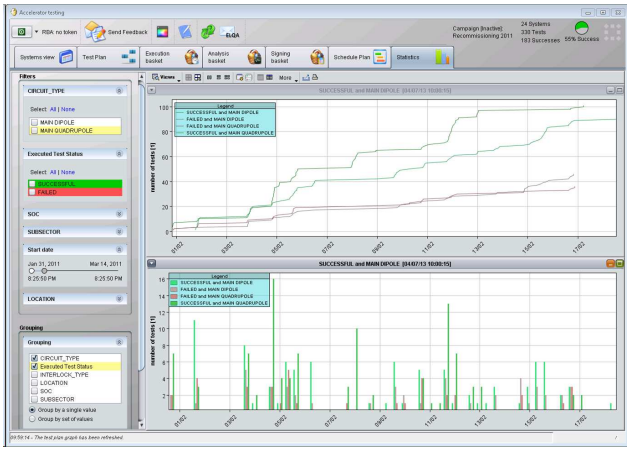

Figure 4: Screenshot of the AccTesting GUI, showing the statistics panel.

The whole system was successfuly used during recommissioning of the LHC circuits after the Christmas stops of 2012 and 2013 and has proved its stability and maturity.

From Sharepoint to AccTesting

In the following, the most importantent concepts of AccTesting, which are required to use AccTesting within the scope of of MPS commissioning, will be briefly sketched.

Currently, AccTesting uses three different 'granularities' in test exectution and tracking:

- The basic building block for a test plan is a *test*. A test is allowed to be executed on one or more system types and can be activated and deactivated per test plan.
- Each test has three *test steps*: Execution, Analysis and Signing. During the execution step actions are performed on the system under test. This means that this is the only time where the system is really blocked. During the analysis step, signals of the system (which were recorded during the execution step) are analysed either automatically or by some external system. The final step is the signing step, which requires human interaction of different experts (depending on the test), who have to verify the outcome of the execution/analysis and sign with their name.
- Each test belongs to exactly one *test phase*. A phase groups tests together and forms the basic building block in the execution sequence. The phases depend on each other. While tests within a phase can be executed in arbitrary order and even if the other tests within the phase are not (yet) successfully analyzed or signed, tests of a dependant test phase can only be executed, if all tests of the phases on which the phase depends were fully successful.

The relation between tests and phases is sketched in Fig. 5.

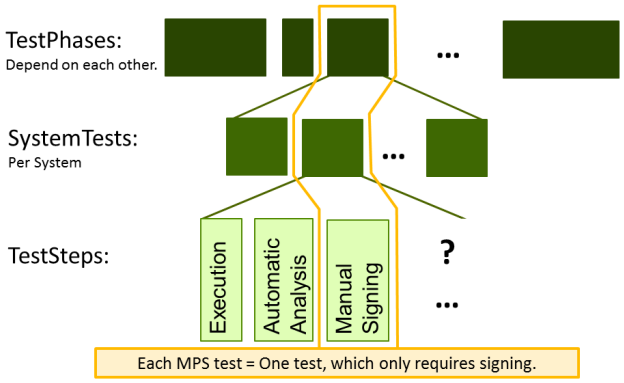

Figure 5: The relation between test phases, tests and test steps in the context of MPS commissioning.

To migrate the information from the old sharepoint site into AccTesting, the following roadmap should be followed:

1. Transform every MPS commissioning step into a test with 'always successful' execution and analysis steps (so called 'sign-only tests'). The tests might be grouped into test phases corresponding to the commissioning plan.
2. Later on, some of these tests can be replaced by automated versions, if possible.

Nevertheless, there are still some additional features which need to be implemented in AccTesting, in order to fully cover the needs of MPS commissioning. These will be desribed in the following section.

Newly required Features in AccTesting

Test Plan Editing Up to now, it was only possible to 'edit the test plan' by direct interacting with the database. Since this is problematic (due to e.g. security, consistency, required expert knowledge), GUI support for performing this task is in preparation. This will be especially needed, as soon as AccTesting is used in a broader field. The test plan for MPS commissioning might have to be adapted quite frequently – at least during the first campaign – with the experience gained. The plan is to provide at least basic functionality in the beginning of 2014 to be able to start with creating test plans (Creating campaigns, enable/disable tests). Extended functionality (Editing of Phases, Barriers and Composite Tests – see following sections) might have to be postponed until later in 2014.

Barriers Currently, AccTesting only takes care about test order and phase dependencies per system. Nevertheless, for MPS commissioning (and possibly for other applications in the future) a more flexible approach is required which allows to relate tests between different systems. The first naive approach would be to extend the concept of phases to a kind of 'global phases'. In the end, this approach turns out to be too strict for the purpose of MPS commissioning, as it would enforce that several tests of different systems have to be done exactly in one global phase. Nevertheless, the appropriate specification would be more like e.g. 'BLM individual system tests have to be done *at some stage before* injecting beam' but not necessarily 'in an individual system test phase' (e.g. there might be a phase 'Powering Tests' between 'Individual System Tests' and 'Injecting Beam'). Therefore, a new concept called *barriers* is proposed for this purpose:

A barrier can be put between two test phases of a several systems. It will allow each system which is affected by the barrier to perform its tests until the barrier point but not beyond. As soon as all the concerned systems reach the barrier point, each of them is allowed to continue with the following tests. This allows to complete the test plan in a very flexible way, while enforcing the required constraints. An example with two barriers is shown in Fig. 6.

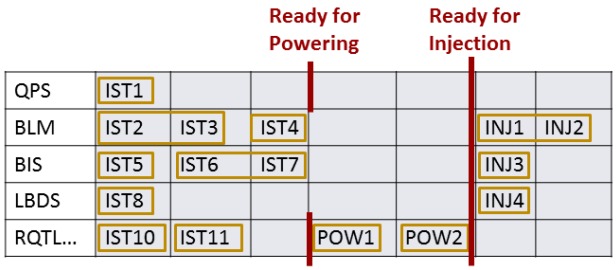

Figure 6: Test barriers in an example MPS commissioning plan. Boxes with brown boarders represent test phases, names within the boxes represent tests and red lines represent barriers.

Composite Tests & System Dependencies Currently one test in AccTesting is assigned exactly to one system. While this approach fits well to the needs of LHC hardware commissioning, the situation for other systems might not be that simple: One system might consist of several subsystems and tests might be formulated in a way that a set of tests on each subsystem have to be completed in order to contribute to the outcome of the test of the composite system. An example could be a test for a BLM crate consisting of one test for each BLM connected to that crate. To model this behaviour an additional feature has to be implemented in AccTesting to allow the definition and the tracking of such so-called *composite tests*.

Another service, which is required by this feature, was put in place recently: The so-called *System Relations Service*. This framework, which allows to plug in different sources of information (so-called 'System Relation Providers'), provides a central service for any kind of software application to query relations between systems. This service is currently embedded in the AccTesting server but

can be extracted to a dedicated server if required. Already now, the service manages information of roughly 17000 systems and 28000 relations between systems.

Automated Analysis In previous hardware commissioning campaigns, most of the signals resulting from test execution were either analyzed manually or by semi-automated tools written in LabView. To unify the approach, a new subproject was started earlier this year which will provide the following components:

- A dedicated assertion language (Java embedded Domain Specific Language - eDSL), which will make it easy for experts to formulate test conditions and necessary related calculations (See Fig. 7).
- A viewer component for the GUI which shows the signals used in the assertions for a test and the outcome of the checks (See Fig. 8).

Also this feature will be useful for MPS commissioning in the future, when automation is applied in the tests. Further extensions are planned, e.g. the usage of different signal sources (Logging Db, Post Mortem, Files) as well as the implementation of more numerical operations on the data. A main concept for this analysis framework is its flexibility to replace implementations of operations at a later stage by more efficient ones (e.g. executed directly in the database), without changing the higher layers (eDSL). Furthermore, distribution of the analysis processing steps on clusters is under investigation, which would allow this framework to become a very fast, horizontally scaling, multi-purpose analysis framework.

```
public class PnoD1 extends AnalysisModule {
    {
        assertThat(I_MEAS)
                .isEqualTo(55.0, AMPERE)
                .withinRel(1.0, PERCENT)
                .at(1, SECOND)
                .before(Occurrence.FIRST, PM_EVENT_TRIGGER);

        assertThat(V_MEAS)
                .isEqualTo(-1.0, VOLT)
                .withinAbs(0.2, VOLT)
                .starting(50, SECOND)
                .after(Occurrence.FIRST, PM_EVENT_TRIGGER)
                .ending(70, SECOND)
                .after(Occurrence.FIRST, PM_EVENT_TRIGGER);
    }
}
```

Figure 7: Example of a script for automated test analysis, written in the dedicated Java embedded domain specific language.

EARLY DETECTION OF FAILURES

While the previous sections were focussing on the improvements of the environment for commissioning the machine protection systems, another aspect of potential improvement manifestated during the previous run: It turned out that many failures were detected rather late during operation, while the problems that led to them could have been detected much earlier. Consider the following example: If a trim is sent from the orbit steering application (YASP) to the LHC software architecture (LSA), then it will be sent directly to the machine. As soon as the power converters ramp the electric current, it might be that one or the other goes out of some interlock limits, for example. This would be detected by a interlock system, which would trigger a beam dump. This dump could have definitely been avoided (if, e.g. the interlock limits would have been taken into account before a real trim in the machine).

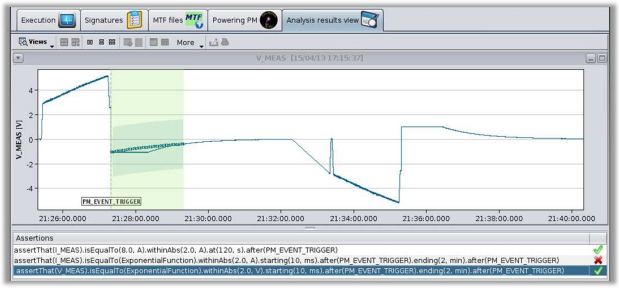

Figure 8: Example display of a result of the analysis of a powering test. The lower part of the window shows the assertions and the upper part shows the signals used in this assertions as well as markers for successful or failed regions.

The natural place to perform such additional checks turns out to be LSA itself, since all trims pass through it, no matter from which application they are sent. After some discussion with the LSA team, the following solution is proposed:

- A first implementation could be put in place using already available mechanisms which are called 'TrimPostProcessor's. A trim postprocessor is invoked any time after a trim is saved into the LSA database, but before the trim is sent to the hardware. By implementing dedicated postprocessors, which would do the check against the interlock limits and throw exceptions if the trim should be aborted, LSA would enforced to perform a rollback on the database, the values would never be sent to the hardware and the application who sent the trim would receive an exception.
- On the longer term, an API which will allow to query the validity of a trim before really executing it, should be provided for the applications.
- Since the incorporation procedure is nothing else than a trim, the described mechanism would also prevent incorporating trims which would trigger a dump somewhere later in the beam process.
- An additional override mechanism might be required for machine development periods.

The following additional changes to LSA could further

improve the security of the LHC operation:

- Currently, only selected methods in LSA are protected from usage without sufficent privileges (RBAC). All LSA methods should be reviewed, if they can do any harm or not, and should then be protected accordingly.
- The cycles which contain the settings for the PcInterlock and the software interlock system should also be protected by RBAC. A first solution could also be implemented by TrimPostProcessors, which evaluate the current RBAC roles.

APERTURE METER

Another tool which was already very useful during previous comissioning phases and will become even more important during the coming ones, is the so-called Aperture Meter. This tool is able to display online the actual aperture limits per beam and per plane over time. A sample screenshot of its main screen is shown in Fig. 9. Furthermore, it can display detailed information about the beam trajectory and plot it together with the aperture model as shown in Fig. 10.

Figure 9: Main Screen of the LHC Aperture Meter. For each beam and plane it shows the distances of the five elements closest to the beam over time.

The current implementation of the aperture meter offers already the most important required functionality: It can follow the operational cycle (optics, beam process, time within beam process) and listens to a selected set of LSA trims to reproduce the best known beam orbit of the beams. Nevertheless, some additional improvements have to be done, to help this application to become fully accepted as an operational tool:

- The user interface has to be improved, so that the operation is more intuitive.
- Performance improvements are required, in particular to improve the startup time (model initialization).

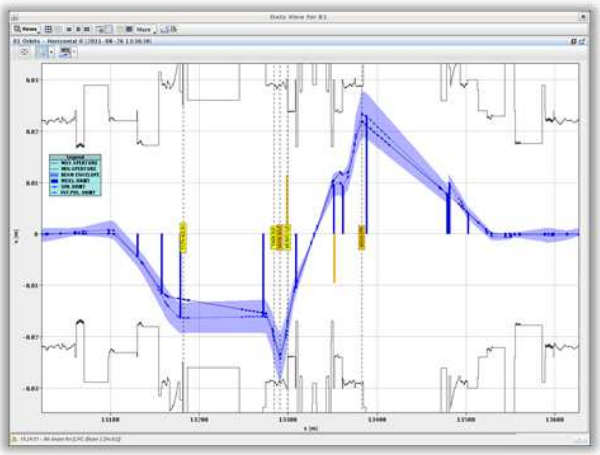

Figure 10: Example live plot of the LHC Aperture Meter. It shows the beam in an IP, together with an envelope of 1σ and the aperture limits.

- Some operational changes have to be better integrated. For example, collimator offsets after allignment or BPM usage information could be read automatically.

In the context of the previous section, the aperture meter itself could be used as an additional source for LSA trim verification (e.g. for collimator movements, collimator hierarchy). For this to work, the aperture meter would have to be implemented in a server, i.e. the functionality would have to be available indpendent if a GUI is running or not. This is not the case at the moment.

REMARKS

Although in the previous section we were discussing many different tools and possible improvements to them, it should be mentioned here that tools do, by no means, solve everything. On the contrary, more important is the development culture and communication during the development of the tools. Currently, software development in the accelerator sector is facing the following challenges:

- Large part of the software manpower goes into maintenance.
- A lot of 'grown' projects exists, partly written by unexperienced programmers (e.g. Students).

To improve the situation, first of all awareness for this problematics has to be raised. Reliability of software is closely coupled to maintainability, which is again equivalent to quality. Quality basically boils down to self explaining code and automated testing. To avoid in the future adhoc software projects, which are often created by unexperienced programmers, it is recommendat that any upcoming student software project is supervised by two distinct persons with different views: One system expert and one software expert (Software 'Mentoring'). Another problem is

that most of the time there is no single person who has the full picture and who can judge what tools are already available, which tools could be extended, or which framework would fit best for a newly required feature. Once again this boils down to communication. Similarly, there is also no single instance (persion, section or similar) with the authority to re-arrange priorities between different software projects. As a result, the limited manpower might not be optimally distributed amongst the projects.

SUMMARY AND OUTLOOK

The main focus of this paper was to elaborated the principal steps which should be taken to improve the commissioning phase of the LHC machine protection systems. We showed how the AccTesting framework could be the workhorse in future commissioning campaigns and we introduced the new features and concepts that will have to be implemented to achieve these goals.

Beyond this, of course further improvements could be envisaged: As soon as a testplan for the commissioning of the machine protection system system is in place, further automation should be discussed. The manual tests can then be easily replaced one by one by automated versions. Further steps could also be e.g. interlocks based on test plans, which would ensure that the tests really have to be performed before operation of the LHC can start again.

Finally, we emphasized that the reliability of a system starts with quality, which is not trivial to achieve and expensive (in time). Nevertheless it must not be reduced by any means. This is especially valid for software related to machine protection and operation, which has to guarantee the safe operation of the LHC and all its subsystems.

ACKNOWLEDGEMENTS

The author wants to thank the whole TE-MPE-MS software section for all their fabulous work and their support. Further thanks to M. Zerlauth, R. Schmidt, J. Wenninger, G. Kruk, V. Baggiolini, G. Papotti and D. Jacquet for their input on this topic.

REFERENCES

[1] B. Bellesia et al., “Information Management within the LHC Hardware Commissioning Project”, proc. of PAC09, Vancouver, BC, Canada.

[2] K. Fuchsberger et al.,“Automated Execution and Tracking of the LHC Commissioning Tests“, proc. of IPAC12, New Orleans, LA, USA.