TESTING FRAMEWORK FOR THE LHC BEAM-BASED FEEDBACK SYSTEM

S. Jackson, D. Alves, L. Di Giulio, K. Fuchsberger, B. Kolad, J. Pedersen
CERN, Geneva, Switzerland

Abstract

During the first Large Hadron Collider (LHC) [1] shut-down period, software for the LHC Beam-based Feedback Controller (BFC) and accompanying Service Unit (BFSU) [2] was migrated to new 64-bit multi-core hardware and to a new version of CERN's FESA [3] real-time framework. This coincided with the transfer of responsibility to a new software team, charged with readying the systems for beam in 2015 as well as maintaining and improving the code-base in the future.

In order to facilitate the comprehension of the system's 90'000+ existing lines of code, a new testing framework was developed which would not only serve to define the system's functional specification, but also provide acceptance tests for future releases. This paper presents how the BFC and BFSU systems were decoupled from each other as well as from the LHC plant's measurement and actuator systems, thus allowing simulation-data driven instances to be deployed in a test environment. It also describes the resulting Java-based Domain-Specific Language (DSL) which allows the formation of repeatable acceptance tests.

INTRODUCTION

LHC operators rely on a feedback system in order to stabilise/correct and adjust the beams' closed-orbit, betatron tune, energy and radial loop [4] during the various stages of the operational cycle from filling to flat-top. The system is triggered at up to 25Hz by input signals from over 1'000 Beam Position Monitors (BPM), along with 6 tune measurements from Base-Band Q (BBQ) [5] measurement systems. The BFC sanitises this data, before calculating the necessary currents to send to various LHC steering dipoles and quadrupoles - The effects of which should be observed in the next iteration of the feedback loop (via the BFSU 1Hz instrumentation layer).

When a new team took charge of the existing codebase, it was apparent that many changes would be required to port the code to new 64 bit hardware, and to migrate to the latest FESA framework. This meant that testing the new software before the LHC start-up would be imperative. Testing using real signals from over 1'500 I/O devices round the LHC would not only be difficult during the machine's shutdown, but would be impossible when the LHC became operational. Consequently it was decided to create a testing framework which could emulate the BFC's input signals, the effect of which could then be observed via the BFSU at 1Hz. Test cases would then be written, for asserting that the BFC and BFSU were reacting correctly to the conditions defined by the test. It was also acknowledged that test developers may not be software experts, and may therefore be uncomfortable learning how to use a potentially complex API. In this context, a descriptive DSL would abstract the testing framework to be easily used by test creators.

Realising the framework, would not only require new testing code, but would entail many changes to the BFC and BFSU themselves, in order to allow instances of the software to be deployed outside their specialised operational hardware. Also, spoofing the BFC's input data would require relaxing of the BFC's data integrity checking mechanisms without compromising security in the operational system.

ADAPTING TO LIFE IN THE LAB

During 2014, the BFC and BFSU were ported to the new 64 bit architecture / FESA3 framework and the team had a release candidate ready for testing. Before any testing framework could be developed however, several changes to the software would be required to allow the software to be tested.

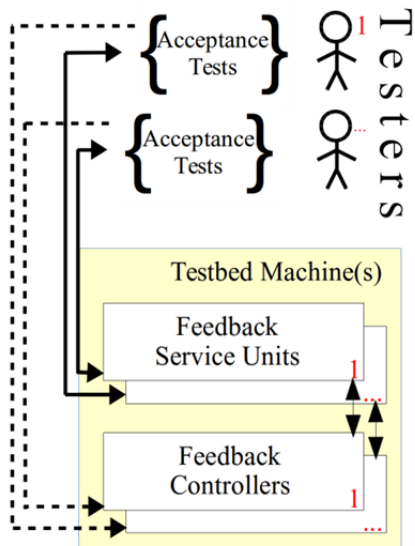

Figure 1: Software adapted so any number of BFC / BFSU pairs can be started, even on the same machine.

The new operational feedback system comprises of 2 multi-core Linux based (with Red Hat's MRG* kernel extension) HP Gen8 Blade machines. In order to assure real-time performance, the BFC is isolated from the LHC control system, with the BFSU acting as a client-facing data / setting proxy. A 2nd Ethernet link is used to transfer (~130Mb / sec) data between the BFC and BFSU

* Messaging Real-time Grid.

ISBN 978-3-95450-148-9

during operational conditions. Reproducing this exotic hardware configuration would be both costly and unnecessary for our testing framework, so the code was modified to allow any BFC instance to be connected to any BFSU on any IP address including loopback (i.e. both on the same machine). Deploying in loopback proved problematic with the original software design, as the BFC and BFSU were both bound to similar network ports, so the 36 existing hard-coded port numbers where converted into offsets with respect to a single configurable base-port. This way, any number of BFC / BFSU unit pairs could be started on the same machine without having any port clashes (see Fig. 1).

The BFC design is different to most of the real-time software developed for the LHC due to its tight real-time constraints. Consequently, the system was compiled and linked on the operational hardware itself, using machine optimized compilations of CERN's Root libraries. This was not ideal for compiling and testing via the new framework, so the system build was adapted to use the standard SLC6 environment. It was assumed that with the latest 64 bit hardware increasing CPU power from 4 core to 24 core machines, real-time constraints would not be compromised.

TEST SYSTEM OVERVIEW

While testing a system would traditionally start with writing unit tests for individual components and then building integration and acceptance tests on top of these fundamentals, the approach taken here was a bit different. Starting with unit tests was problematic, because classes on the BFC have a high inter-dependency. Our main goal was to bring the system (focussing mainly on the BFC) into a test harness in order to observe and verify the global behaviour while gaining confidence for future changes. To accomplish this, it became clear that the BFC's I/O channels (UDP packets, described later) could already provide us with a clean way to inject our tests. This choice has the following advantages:

- Minimal changes required in the BFC.
- UDP is language independent; we were free to choose any language for the test system.

Given the option of choosing any language, we decided to use Java as the implementation language for the testing framework, for the following reasons:

- Java is widely used in the LHC environment, so more people will be able to easily use and extend the framework providing more tests.
- Interactions with other parts of the control system (e.g. settings management) are natively possible in Java. This allowed us to easily write tests involving operational settings and/or even verify consistency of such settings with the framework.
- It allowed us to gain first experience in the behaviour of Java in a real-time environment.

An important aspect taken into account in the design of the testing framework was the fact that tests of different layers would be required. We identified the following types of tests:

- **FESA mechanics**: e.g. asserting that setting a value in one FESA property has the desired effect in another. Despite the fact that this functionality is only for interfacing with the control system, these tests turned out to be the most frequently required, due to the complexity of the BFSU's API.
- **Communication**: e.g. send some predefined values for an orbit and check if the values are correctly processed through the layers.
- **Control loop behaviour**: e.g. send a constant orbit verifying the resulting corrections. From an operational viewpoint these are the most interesting tests, as they highlight instabilities and allow error predictions.

These different abstraction layers are reflected in the testing framework's DSL as described later.

PLANT VIRTUALISATION

Without input signals the feedback system does nothing apart from reporting timeouts. To operate correctly, the BFC needs the following (see Fig. 2):

- **BPM packets**: Sent at 25Hz from the BPM systems, they contain position measurements (~500 values per beam and plane).
- **BBQ packets**: Sent between 1 and 100Hz from 3 BBQ systems, they contain the measurement values for the tunes (1 value per beam and plane).
- **Orbit trigger**: Hardware trigger at 25Hz, which starts orbit averaging in the BPMs, and also data collection in the BFC. On reception of this trigger, the BFC opens an acceptance window (~10 ms), in which arriving packets are processed for one feedback iteration.

In order to test the system, the testing framework must virtualise these BPM, BBQ and Orbit Trigger input signals as well as capture the output signals (to avoid sending to the correction magnets). Luckily, most of the input signals already arrive via UDP packets, so spoofing the data was feasible. The only exception in the new feedback system is the Orbit Trigger which arrives via a cable from the Beam Synchronous Timing (BST) [6]. To facilitate the spoofing of this input signal, code was modified in the BFC, and the trigger's interrupt was handled in a dedicated thread which delivered the trigger into the BFC's main loop via a UDP packet. Although this added a small jitter to the trigger, the additional overhead of passing through a local UDP port was deemed acceptable.

All the operational I/O systems communicating with the BFC are written in C++, and the UDP packets' content are declared as C structs. Our chosen language for the test however was Java, so these structs had to be emulated by Java classes which could be de-serialised / serialised to / from a stream of bytes which mimicked exactly the structs from the real systems. Also, the contents of our packets had to adhere to the strict data

ISBN 978-3-95450-148-9

quality rules specified by the BFC, including information used for diagnostics. Even if we were not interested in testing things such as temperature from our framework, this diagnostics data must be within sane limits.

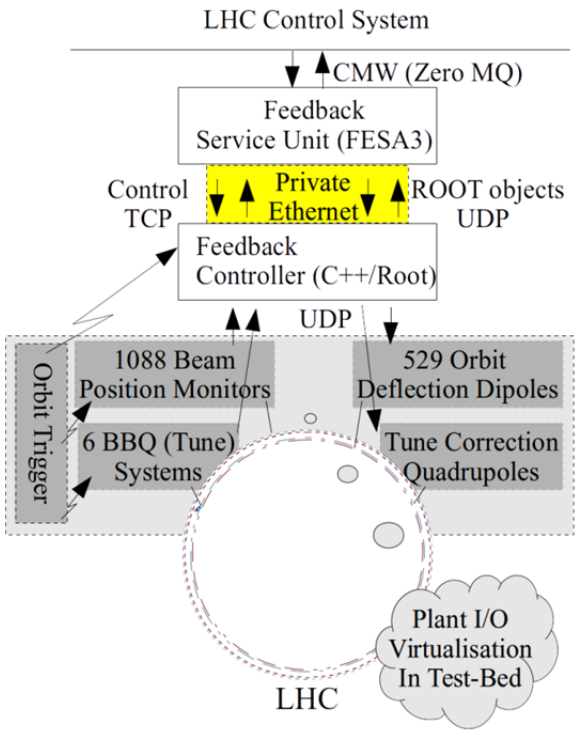

Figure 2: Framework must virtualise LHC I/O signals and decouple from private Ethernet dependency.

TEST FRAMEWORK ARCHITECTURE

Figure 3 shows the core components of the testing framework and their different layers as follows:

- The main work and internal logic is done in the service layer. It combines data from the lower layers and provides convenient methods for the upper layers for dedicated tasks (e.g. 'create an orbit with certain properties').
- The lowest *backend* layer contains various adaption / delegation classes for subscribing to FESA devices, creating simulated orbits, reading settings, creating UDP packets, etc.
- From a functional perspective, the service layer would be sufficient to write the tests. However, to make tests more expressive and readable by non-programmers, we followed domain driven principles and put a layer of a Java-embedded Domain Specific Language (eDSL) on top. It was implemented as a fluent Java API, using method chaining as the main syntax concept. The starting points for all the fluent clauses are grouped by topic in classes (e.g. OrbitCreationSupport for clauses which create different orbit types).

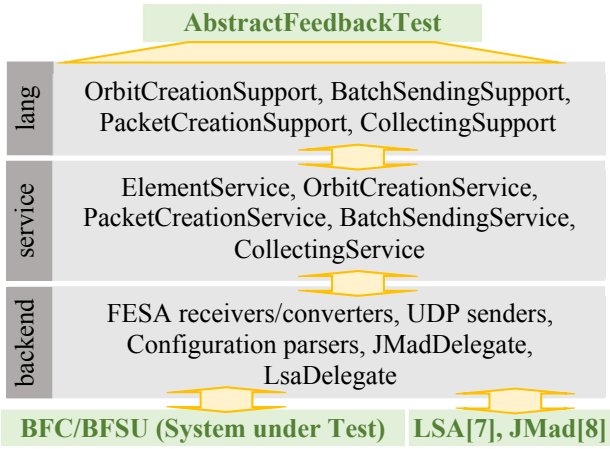

Figure 3: Overview of the testing framework architecture.

As a framework to execute our tests, we chose the JUnit framework, which simplified test creation and report generation as well as using well established matching libraries (hamcrest, assertj) to formulate assertions. The framework provides an abstract class, *AbstractFeedbackTest*, which can be used as a base class for all tests. It is preconfigured with instances of each support class and provides delegation methods to them, so that inheriting tests can formulate fluent clauses without instantiating any additional objects.

THE DOMAIN SPECIFIC LANGUAGE

To get a general impression of the different features of the testing framework and the eDSL in particular, we will show some selected examples. Since the execution of the tests is based on JUnit, the skeleton of a test will always look something like this:

```
public class ExampleFeedbackTest extends AbstractFeedbackTest {
    @Test
    public void assertThatSomeValueIsCorrect() {
        /* Test Code goes here */
    }
}
```

Most tests involve communicating with the BFC, so the language provides the means to construct packets of a particular type and send them:

```
Burst burst = dummyBpmPackets(NUMBER_OF_PACKETS_PER_BURST,
                              constant("X"), BEAM_POSITION);
prepareSendingOf(burst).during(5, SECONDS).
                every(40, MILLISECONDS).andStartAndAwait();
```

A burst represents a set of packets which simulate the state of the LHC at a moment in time (i.e. one orbit). A more meaningful test would be to construct an orbit with all zero values and send it (this time using defaults for duration / period, spawning the sending in the background, checking the publishing from the BFSU and then awaiting the termination of the sending):

```
prepareSendingOf(zeroOrbit()).andStart();
MultibeamOrbit acquiredOrbit = from(MultibeamOrbit.class).
                                skip(1).and().awaitNext();
awaitLastSending(); /* assertion here */
```

Creation of other orbits is also supported. e.g. Constant single outliers or (artificial) flat orbits with constant values:

ISBN 978-3-95450-148-9

```
MultibeamOrbitReading first =singleOutlierOf(1.2f, MILLIMETER).
                        at(Bpm.ofName("BPM.24R2.B1"), HORIZONTAL);

MultibeamOrbitReading second = sameValueOrbitOf(4, MILLIMETER);
```

Finally, the language also provides a convenient way to retrieve and manipulate FESA properties and fields.

```
String opticsName = from(OPTICS).doGet().getActiveOpticsName();

to(ACTIVE_OPTIC).partially().setFetchOFC_object(true);
```

This last feature turned out to be very useful for other applications and will be extracted in the near future to a dedicated package to facilitate its re-use by others.

RESULTS

Within minutes of first beam, the new feedback system was supplying operators with orbit data, as well as successfully applying orbit and tune feedback. When operators tried to load different reference optics however, the new software didn't behave as expected. Despite testing most of the basic system functionality, the framework didn't have a dedicated acceptance test for reference optic changes, but the team were able to quickly reproduce the phenomenon in the testbed, solving the problem without further disturbing operations. Another problem not covered by the tests involved a rogue BPM system sending corrupted timestamps which caused the Software Interlock System to dump the beam several times. Again, the team reproduced the phenomenon in the testbed, and quickly supplied a patch to fix the problem operationally.

A weakness of the framework was highlighted during the first few months of operation, when problems with the delivery of timing events occurred during optics reference changes. In this case, the testbed was unable to reproduce the problem unless an accompanying test framework for the timing event delivery was also available. In the long term, it is hoped that such a framework will become available, but in order to solve this issue, the operational system had to be used to diagnose and solve this problem.

FUTURE PLANS AND OUTLOOK

The acceptance test coverage is still relatively low, so the team plans to continue developing new tests for existing functionality as time allows. However, any new feature added to the BFC or the BFSU must have accompanying acceptance tests if possible.

During the next 12 months, the team plan to implement a new streamlined BFSU API which will use new features now offered by the latest FESA3 framework. The delivery of this new API will be progressive, and released in parallel with the existing API. In order to assure that the new API is functionally equivalent, the plan is to duplicate the existing tests and adapt them to the new API. A third set of tests will then assert that the results of the tests with the old API and new API are equal.

Presently, the output of the BFC in test conditions is muted. There is an ambitious plan to capture the BFC output (UDP packets destined for the corrector magnets) within the acceptance test, and combine it with the input signals for the next iteration. With this in place, we effectively close the feedback loop within the testing framework.

There are also plans to use the experience from this testing framework to test other systems. Several systems such as the BBQ tune measurement system and the LHC beam-loss monitoring system will also require porting to the new FESA3 framework, and will therefore benefit from offline testing. Whilst the scope of tests on these systems will be less than for the feedback system (they rely on dedicated hardware), basic I/O tests can still be performed. In the long term, these systems could also benefit from more extensive acceptance tests, if like the feedback system, the systems are re-written to accommodate simulated data in place of acquisition cards for example.

CONCLUSION

The testing framework enabled the new software team to understand the code they inherited, and provided functional documentation in the form of test assertions. The tests highlighted many bugs which would probably have gone un-detected until the LHC start-up; bugs which would have hindered the operators during a very critical period. Importantly, the testbed has also given the new software team the confidence to roll out several changes to the software during 2015, citing the testbed assertions as confirmation that the new release should be acceptable. Of course the testbed will never guarantee that a new release is bug-free, but the confidence is certainly increased.

The resulting framework has proved that one of the most complex pieces of software in the LHC can be tested offline, and should serve as an inspiration to develop other test frameworks for other critical systems at CERN.

ACKNOWLEDGMENT

The team would like to thank Jörg Wenninger from the Operations group, Vito Baggiolini from the Controls group and Markus Zerlauth from the Machine Protection and Electrical Integrity Group for their considerable contribution of time and advice during the project.

REFERENCES

[1] Lyndon Evans and Philip Bryant, "LHC machine", JINST, 3 (2008), p. S08001.

[2] L. K. Jensen et al, "Software Architecture for the LHC Beam - Based Feedback System at CERN", ICALEPCS'13.

[3] M. Arruat et al., "Front-End Software Architecture", ICALEPCS'07.

[4] R. Steinhagen "Real-Time Beam Control at the LHC", PAC'11.

[5] M. Gasior et al., "Advancements in the Base-Band-Tune and Chromaticity Instrumentation and Diagnostics Systems during LHC's First Year of Operation", DIPAC'11.

ISBN 978-3-95450-148-9

[6] D. Domínguez et al., "An FPGA Based Multiprocessing CPU for Beam Synchronous Timing in CERN's SPS and LHC", ICALEPCS'03.
[7] G. Kruk et al., "LHC Software Architecture [LSA] -- Evolution Toward LHC Beam Commissioning", ICALEPCS'07.
[8] K. Fuchsberger et al., "Status of JMad, the Java-API for MADX", IPAC'11.

ISBN 978-3-95450-148-9