Summer Student final report
Due on Thursday, September 26, 2013

Jakub Guzik

Supervisor:
Ivan Romera Ramirez
Abstract

During my time spent at CERN I worked under the Technology Department of CERN, in the Machine Protection and Electrical Integrity (MPE) Group. The MPE Group supports LHC operations and maintains state of the art technology for magnet circuit protection and interlock systems for the present and future accelerators, magnet test facilities and CERN hosted experiments[1].

As a member of Magnet Powering Interlocks & Software (TE-MPE-MS) section I was involved in three different projects and used not only CERN developed tools like FESA Framework, but also open source C++ frameworks, Google Test and Google Mock. I had a chance to work with Programmable Logic Controllers and real-time devices known as Front End Computers. I was part of a software developer team, and familiarized myself with the Scrum agile software development methodology.

The description and results of my work are presented in three parts of this report. Each part describes a separate project created during my participation in the CERN Summer Student Program 2013.
Contents

1 FESA interface for Access Powering Interlocks 4
   1.1 Introduction ................................................. 4
   1.2 Implementation ............................................. 4
   1.3 Tests ....................................................... 5

2 Timing alignment issues in FMCM FESA class 5
   2.1 Introduction .................................................. 5
   2.2 Time misalignment problem .................................. 5
   2.3 Test environment ............................................. 6
   2.4 Detected problems & solution ............................... 6
      2.4.1 One second problem ..................................... 6
      2.4.2 Incorrect date problem .................................. 6
   2.5 Tests ....................................................... 7

3 Migrating the CIBM FESA class and introducing unit testing 7
   3.1 Introduction .................................................. 7
   3.2 Unit testing .................................................. 7
   3.3 Google Test .................................................... 7
   3.4 Google Mock ................................................... 8
   3.5 New software architecture ................................... 8
   3.6 Summary ..................................................... 9

References 10
1 FESA interface for Access Powering Interlocks

1.1 Introduction

The first project was a part of the modifications that are proposed in the document Change of the Interlocking of Powering and Access Systems (EDMS 1246780), this involved changing the connections between LASS and SIS from currently used TIM system in order to improve the dependability of the existing interlocks involved in the protection of CERN personnel. The Powering permit for the eight LHC sectors will be represented as Access states, which will be collected from additional PLC devices by using a FESA class, and then republished to CMW. SIS will subscribe to those access states. One can read more about proposed changes in the mentioned document. A new layout of the interlocking of access conditions with powering is shown in Figure 1. Red arrow marks a part of the new system which is described in this report.

![Figure 1: New layout of the interlocking of access conditions with powering](image)

1.2 Implementation

The FESA 3 class design was generated by the ietool3, which was created and developed by the IEPLC project team within CERN. The Class was later modified to fit new system needs described in the Introduction section. The FESA 3 project provides tools to support creating a project skeleton and default actions, but every project created by these tools should be
customized to serve one’s needs. Therefore, a real time action was modified to check every
second for access condition changes and update them when changes are detected.

1.3 Tests

The New FESA class was tested with Siemens SIMATIC S7 PLC device and Siemens software
in TE-MPE-MS section’s laboratory. Tests have shown that the class works without any
problems. After testing, project documentation was written to describe how to maintain the
FESA class in case of future device changes, such as renaming, relocation, etc.

2 Timing alignment issues in FMCM FESA class

2.1 Introduction

The Fast Magnet Current Change Monitors (FMCMs) are designed to monitor and detect
fast current changes in normal conducting magnets in critical parts of the LHC and its
transfer lines. When a FMCM detects that a change in the magnet current can have an
impact on the beam orbit, a beam dump request will be sent to the Beam Interlock System.
These current changes are typically caused by powering failures.

In the current implementation, FMCM devices instantiate the FMCM FESA (obsolete 2.10
version) class written in 2006, which reads data from devices and publishes them to the
Controls Middleware (CMW). The most important data acquired by the FMCM is the Post
Mortem data, which contains useful information such as voltage and current recordings
required to validate the correct functioning of the system after a beam dump. Global PM
events are triggered via the LHC timing system and when such an event occurs data from
all connected devices are collected.

2.2 Time misalignment problem

FMCM devices use an internal clock to time stamp any events that occur as well as any useful
PM data. In order to improve the accuracy of the local clock and to remain synchronised
with the LHC General Machine Timing (GMT), a timing interface has been implemented.
There are two signals involved in the synchronization process: the UTC time which indicates
the absolute UTC with a periodicity of 30 seconds and the pulse per second (PPS) which
has a periodicity of 1Hz with and a resolution of some nanoseconds.

The problem that occurs is that sometimes FMCM devices lose synchronization with re-
spect to the LHC GMT. Two misalignment problems were detected, the first one occurred
randomly and usually it was only a 1 second mismatch, the second one was that the time
misalignment was completely random.
2.3 Test environment

A proper test environment was created in collaboration with other members of the TE-MPE group, in order to detect the source of the problem. It consisted of a Lab View program which was triggering a fake PM Event every minute to check whether the timestamp of the corresponding PM data of the FMCM matched the expected time. Special python scripts were written in order to help with analyzing FESA FMCM class output data, and detect the frequency of observed misalignments.

2.4 Detected problems & solution

2.4.1 One second problem

The first detected problem was a one second misalignment problem, and it was easy to track and to correct. The problem was occurring because of a wrong UTC time update approach. Time was sometimes updated at the beginning and sometimes at the end of a second, and because there are frequent delays in transmission, UTC time update were sometimes received too late. This is why time was shifted one second in the past. To correct this problem, a simple algorithm was proposed and implemented within one method, which always updates time at the beginning of every second. Currently the update window is set to 0.06 of second. The pseudo code of the proposed solution is shown in the listing below.

```c
long UTCcommand::ReturnUTCTime(){
    time=getCurrentTime()
    updateWindow=0.06 sec
    while(time.nanosecondPart>updateWindow){
        sleep(10 microsec)
        time=getCurrentTime()
    }
    return time.secondPart
}
```

2.4.2 Incorrect date problem

The second problem detected was harder to find, because it was occurring even more randomly. The FMCM device uses the whole ASCII table to transfer messages, even the NULL sign. It is not the sign for the end of a message, as it should be in the normal case, but also as other signs it carries information. Every command message in the FMCM specification has the same length of 20 bytes (characters), but instead of being fixed to 20 bytes, the C function `strlen()` was used to count the length every time it was composed. Every cstring in C and C++ is meant to be a NULL ended string, so sometimes `strlen()` function just counted the length to the first NULL sign and ignored the remainder of the message. This
error in implementation was present in every real time action of the FMCM FESA class. To correct this problem every `strlen()` function was replaced by a fixed 20 bytes size of a message.

2.5 Tests

The Proposed changes were tested for a whole week in a previously created test environment. PM Event triggering time was set to 30 seconds, 1 minute and 10 minutes. In every case there were no errors observed previously, and all detected problems were fixed.

3 Migrating the CIBM FESA class and introducing unit testing

3.1 Introduction

The CIBM is the central element of the Beam Interlock Controller. It is based on VME-bus technology since it is the standard adopted by CERN for fast interlock devices. The CIBM Software interface was developed with the FESA 2.10 framework. The current 3.0 version of the framework is incompatible with the old one. There are no tools for automatic code migration, which is why every application using old version should be migrated manually.

3.2 Unit testing

The MPE software group decided that during code migration, unit testing should be introduced to the application. It is a testing method widely used in programming languages to test individual parts of code in a stand-alone way. For example, individual methods from a tested part of the code could be invoked and tested against special, boundary cases, and other fixed situations. Google Test and Google Mock frameworks will be used for testing in the designed FESA class.

3.3 Google Test

Google Test is a framework for writing C++ tests on different platforms. It supports a rich set of assertions, fatal and non-fatal failures, many options for running the tests, and XML report generation. Software testing is not as popular in C++ as in Java applications, but various significant projects are using Google Test (for example Chrome Browser, LLVM compiler)[2]. Google Test is easy to use and well documented.
3.4 Google Mock

One of the main ideas of testing is to test developed software separately from third party devices and libraries. A mocking framework should be used to mock objects that are depending on third party software and hardware. Google Mock is a default mocking framework for Google Test. It helps with creating mock classes and defines their behavior depending on received parameters, and other circumstances[3]. It is inspired by similar projects jMock, EasyMock and Hamcrest.

3.5 New software architecture

Google Mock requires specific software architecture to implement mock classes without modifying operational source code. The tested part of the code, in this case the CIBMBoard class, has to communicate with third party libraries through special delegates, as shown in Figure 2. They are purely virtual classes, so in test case, they are implemented as mock classes, as shown in Figure 3.

![Figure 2: New software architecture in operational case](image-url)
3.6 Summary

The new software architecture was proven to work in both test and operational cases. However, there are several unresolved problems, mostly with using elements of the FESA framework with Google Test and Google Mock framework:

- Files with test `main()` functions have to be in one of the `.cpp` files below class implementation, otherwise tests won’t compile,
- Some parts of the FESA Framework used with tests (for example `FesaIOException`) cause a Segmentation Fault error,
- In the `Makefile.specific`, one has to double specification of included, otherwise tests won’t compile.

All detected problems were forwarded to the FESA Support Team. A short presentation was made and presented in front of MPE software group to summarize and evaluate the possibility of introducing software testing in FESA classes.
References

