Kernel Design of a Flexible Software Framework for Magnetic Measurements at CERN

Pasquale Arpaia\textsuperscript{1,2}, Mario Luca Bernardi\textsuperscript{1}, Luca Bottura\textsuperscript{2}, Marco Buzio\textsuperscript{2}, Laurent Deniau\textsuperscript{2}, Giuseppe Di Luca\textsuperscript{1}, Vitaliano Inglese\textsuperscript{2,3}, Juan Garcia Perez\textsuperscript{2}, Giovanni Spiezia\textsuperscript{2,3}, Louis Walckiers\textsuperscript{2}

\textsuperscript{1}Department of Engineering, University of Sannio, Corso Garibaldi 107, 82100 Benevento, Italy. Ph: +39 0824 305804-17, Fax: +39 0824 305840, E-mail: \{arpaia-mlberrnarl-dilucca\}@unisannio.it
\textsuperscript{3}Department of Electrical Engineering, University of Naples Federico II, Via Claudio 21, 1-80125, Napoli, Italy.

Abstract – The kernel design of the Flexible Framework for Magnetic Measurements (FFMM), under development at the European Organization for Nuclear Research (CERN), is described. The design aims at satisfying the critical scenario of short-term requirement variability for the small batches of magnets after the commissioning for the Large Hadron Collider (LHC). The experimental test results of software flexibility and scalability are illustrated for a case study of superconducting magnets in warm and cold on-field conditions.

Keywords – Accelerator measurement systems; Magnetic variable measurement, software flexibility.

I. INTRODUCTION

The large effort required at European Organization for Nuclear Research (CERN) for the test of the superconducting magnets for the Large Hadron Collider (LHC) has led to specific developments of software tailored to the needs of repetitive measurements. The main focus was on quality control and fast production test operations on a park of well defined instruments and measurement conditions. Given this objective, software flexibility and reusability have naturally received lower priority. The end of the series tests on the LHC magnets has marked a change in these requirements. In particular, the need for a uniform platform to span all magnetic measurement applications, increasing the flexibility of the measurement stations, and facilitating changes in the hardware configuration and measurement conditions, arises.

Conceptual work in this direction has started in cooperation with the University of Sannio, by analyzing carefully the state of the art, at commercial level [1], at research level [2]-[3], and inside the CERN [4]-[5]. CERN and University of Sannio prototyped a Flexible software Framework for Magnetic Measurement (FFMM) in order to satisfy the new magnetic measurement requirements highly evolving during time. The project, preliminary presented in [6] as an object-oriented framework, was further improved in [7], by detailing its basic ideas and architecture, by exploiting an aspect-oriented approach, and by presenting some experimental results obtained by means of a demo bench. Aspect-oriented concepts were widely discussed in specific papers, where a new approach to fault detection in automatic measurement system was proposed [8]-[9]. Finally, aspect-oriented approach was extended also to the software synchronization in the measurement procedure [10], where the first results of on-field tests by means of the rotating coils technique were also provided.

However, only the basic approach and the conceptual project to cope with the main goals were faced, while a comprehensive design of the framework kernel was still missing.

This paper aims mainly at filling in this gap. In the following Sections, the FFMM kernel design, the implementation, and the experimental results of the on-field flexibility tests for a prototype based on rotating and fixed coils techniques [11] are described.

II. PROPOSAL

The FFMM is a software framework for magnetic measurement applications based on Object Oriented Programming (OOP), and Aspect-Oriented Programming (AOP) [12]. In particular, FFMM aims at supporting the user in developing test software maximizing quality in terms of flexibility, reusability, maintainability, and portability, without neglecting efficiency, vital in actual test applications. Moreover, the requirements for a wide range of magnetic measurement applications, such as needed for the test of superconducting magnets for particle accelerators, have to be satisfied.

The basic ideas and the architecture (Fig. 1) of FFMM are discussed in [6], [10]. In the following, the FFMM kernel design is described. In particular, a logical view of the architecture is illustrated, by detailing the most important classes, their organization in service packages and
subsystems, and the organization of these subsystems into layers.

The architecture of the FFMM kernel, the Scheme, is shown in Fig. 2. From bottom to up, the main three layers of the framework are shown. Operating system isolation layer is omitted and it will be described in supplementary specifications covering portability issues. In the bottom layer, all the basic services, needed to implement high level logic are placed. This layer includes subcomponents for environment abstraction, memory management, error handling, filesystem abstraction, processes and threads handling. It also defines abstract communication services to high-level components within the FFMM layers, in order to extract data from actual devices and external interfaces, by allowing the exchangeability of the communication mechanisms without incurring into performance penalties. The interface |CommunicationBus is used to send and receive data to/from components in an abstract way. Concrete implementations of such interface are required to handle specific communication devices.

The middle (core) layer includes several packages exposing main functionalities related to components (in particular measurement devices), event handling infrastructure, fault detection, and logging. In the design of the event handling architecture, a variant of the Observer design pattern [13] was used in order to keep synchronized the state of cooperating components (e.g. virtual devices). The Observer enables one-way propagation of changes: one publisher notifies any number of subscribers about changes of its state.

The FaultDetector (Fig. 2) intercepts malfunctions and errors in the measurement systems. Its architecture is based on a fault detection subsystem and a fault notification subsystem. The analysis of several state-of-the-art measurement systems highlighted that fault detection is scattered all over different hierarchies, mainly with reference to devices hierarchy. For this reason, the FaultDetector can be considered as a cross-cutting concern, and an experimental release was conceived according to an aspect-oriented approach, such as detailed in [8].

Logging facilities are also provided at this level of the architecture (Fig. 2). Data can be stored in a text or binary file. In any case, the final destination of the logged messages has to be kept decoupled with the format of the messages themselves. With this aim, two different responsibilities arise, independently varying on each other: logged message formatting, and logged message recording. The formatter does not take care about where the message is recorded, and the recorder does not care about the format of the message. Therefore, the Logger class implements the Strategy design pattern [13]: the concrete logger can be configured with the right formatter and the right recorder keeping them decoupled.

Based on the services provided at this level of architecture, the measurement layer (top of Fig. 2) implements a minimal but extensible infrastructure, based on the TestManager class, in order to handle and to perform measurement session.

For the measurement layer, two main features are needed:

- a test session director, the TestManager, encapsulating the user-script (Fig. 1) and executing it in a controlled environment. Within the user-scripts, core services are made available to the user in order to implement its measurement process;
- the capability of creating groups of data acquisition tasks to be synchronized to well defined events (e.g. start and stop, or device events). In the FFMM framework, the component realizing this high-level software synchronization of data acquisition is the Synchronizer (Fig. 2) [10].

The TestManager organizes the test on the basis of the UnitUnderTest, the Quantity to be measured, the measurement configuration, and the measurement procedure. It has an association with the Devices (software representation of the measurement devices).

The VirtualDevices can be controlled remotely by PC through a CommunicationBus. The Synchronizer manages the software temporization in the measurement procedure.

It is an application of a Composite design pattern [13], which allows basic acquisition processes to be composed together as sequences or parallels of acquisition. Such as the abovementioned FaultDetector, it can be considered a cross-cutting concern, since it is transversal to many software modules. Therefore, in an experimental version, the Synchronizer and the FaultDetector are encapsulated in Aspects according to AOP approach. As a consequence, the policy for managing synchronization actions and faults can be extrapolated from the single modules and handled separately. In this way, further modifications will affect only these two components, without any need for code changes in all the modules related to the fault detection or to the synchronization.

The Scheme was implemented in C++ by individuating the most suitable design patterns for the different phases of the measurement procedure [10]. In the preliminary release, the C++ compiler was also exploited in order to provide the features of Script Checker and Builder. The low-level
code, directly related to the hardware components, was structured according to the most suitable pattern in order to assure efficiency and reliability. The FFMM is based on a single-process instance application comprised of several threads allocated to one or more components in various ways (Fig. 3). The main thread is the main thread (unique) of the FFMM hosting process. It initializes FFMM services and configuration and starts the execution of the user script for the running TestManager. Depending on components that will be created, other kinds of threads can be spawned within script execution. Virtual Devices can spawn internal threads (DeviceThreads) in order to handle polling tasks or critical data transfer in order to meet transfer rate constraints. In the case of interrupt-capable software drivers, this can be not necessary. Logging components, depending on the communication speed and on the complexity of the formatting facilities, can spawn internal threads to meet performance requirements (Logging Threads).

Event handling between capable components can be configured with different multi-threading policies (Event Dispatcher Threads).

Figure 2. Architecture of the Scheme.

Figure 3. Mapping of FFMM processes and threads.
III. EXPERIMENTAL RESULTS

The FFMM platform was aimed at easily producing the software for new magnetic measurements applications. This capability was tested experimentally by performing measurements exploiting different techniques, by changing the devices employed and their settings. The first on-field tests were carried out at CERN SM18 laboratory for measurement applications based on rotating coils [11], one of most accurate techniques for superconductive magnet testing. A coil turns into the magnet under test and its output signal is proportional to the flux derivative, according to Faraday’s law. The coil signal is integrated in the angular domain by means of the output pulses of an encoder mounted on the rotating coil shaft. A Fourier analysis of the flux finally yields the multi-poles of the magnetic field generated by the magnet under test. Subsequently, measurements by means of fixed coils technique were also performed. This technique, used above all for detecting fast field changes, aims at measuring the main field component, rather than higher-order harmonics, obtained from the integration of the voltage induced on a fixed coil by a variable magnetic field. FFMM was therefore required to handle different hardware configurations, in order to measure the magnetic field of the main LHC dipole through the abovementioned techniques. Namely, the devices used for the tests on a LHC superconductive magnet are:

- CERN FDI (Fast Digital Integrator), a CERN proprietary PXI board general-purpose digitalization board, configured for the coil signal acquisition and numerical integration [14];
- CERN Encoder board, a CERN proprietary PXI board, for managing the encoder pulses and feeding the trigger input of a digital integrator;
- TRU (Twin Rotating Unit) and MRU (Micro rotating Unit) [15]-[16], consisting of a motor turning the shaft and an angular encoder that measures the coil rotation;
- Digital Multimeter KEITHLEY 2000, to measure the current in the magnet, read by a high precision Direct Current-Current Transformer (DCCT);
- MIDI Ingenieric motor controller, driving the motor which turns the twin rotating unit;
- EPOS 24, driving the motor which turns the Micro rotating unit;
- Heinzinger PTN 135-20 Power Supply, to feed the magnet with a constant or variable current, for rotating or fixed coils, respectively.

The considered use-cases include three main sets of tests:

- measurements of main dipole field with 6 FDI, a standard coil shaft, and the TRU: (i) a single measurement with one FDI and one turn of the rotating coil, (ii) the standard algorithm “washing-machine” (forward/backward rotation), (iii) only forward, and (iv) only backward rotation;
- measurements of the main dipole field with 6 FDIs, the new rotating coil shaft, and the MRU: (v) the standard algorithm “washing-machine”, (vi) only forward, (vii) backward rotation, and (viii) continuous rotation;
- and (ix) measurements through fixed coils.

Measurements through rotating coils were performed both at cryogenic (4.6 K, 4.0 kA supply current) and room temperature (298 K, 10 A supply current). In Fig. 4, an

![Figure 4](image-url)  
Figure 4. Normal and skew components of the magnetic field at warm condition, rotating coils algorithm with forward (a) and backward (b) rotation.

![Figure 5](image-url)  
Figure 5. Fragments of user script for rotating coils algorithm with forward (a) and backward (b) rotation.
example of the normal and skew harmonics of the magnetic field, obtained by Fourier analysis of the flux at warm conditions, is provided for both forward (a) and backward (b) rotations.

In Fig. 5, for the same example, two fragments of the corresponding measurement scripts are shown. The former (a) is relative to the forward rotation, and the latter (b) to the backward rotation. The comparison of the fragments highlights the light modification of the script in order to reconfigure the measurement application.

The tests results proved that the FFMM’s tools assure a high degree of software flexibility and reconfigurability, without increasing the complexity of the script. Thus, a small effort is required to the user in order to set up new measurement applications or to tailor existing ones.

IV. CONCLUSIONS

The kernel of a software framework for magnet measurements was designed and implemented. The prototype proved the feasibility of a flexible and reusable system to face the new test requirements arising after the production series of the LHC magnets.

In particular, the user is supported in developing software maximizing quality in terms of flexibility, reusability, maintainability, and portability under LINUX and Windows operating systems, without neglecting efficiency, vital in test applications.

The kernel was implemented as a proof demonstrator on a prototype of magnetic measurement bench at CERN laboratory SM18, based on rotating and fixed coils techniques for the harmonic analysis of magnets. The proof demonstrator verified the feasibility of the proposed key concepts and the related architecture. Preliminary results show that the platform can be easily tailored by a test engineer to manage different magnetic measurement applications, and thus encourage to finalize the FFMM design and implementation.

ACKNOWLEDGEMENTS

This work is supported by CERN trough the agreement K1322 with the Department of Engineering, University of Sannio, whose supports authors gratefully acknowledge. Authors thank F. Cennamo for his useful suggestions, as well as Giancarlo Golluccio and Giuseppe “Giuseppi” Montenero for their collaboration.

REFERENCES