FRONT-END SOFTWARE ARCHITECTURE

Michel Arruat, Leandro Fernandez, Stephen Jackson, Frank Locci, Jean-Luc Nougaret, Maciej Peryt, Anastasiya Radeva, Maciej Sobczak, Marc Vanden Eynden
Accelerators and Beams Department, CERN, Geneva, Switzerland

Abstract

CERN's Accelerator Controls group launched a project in 2003 to develop the new CERN accelerator Real-Time Front-End Software Architecture (FESA) for the LHC and its injectors. In this paper, we will report on the status of this project, prior to the imminent start-up of the LHC. After describing the main concepts and components of the infrastructure, we will present how we have capitalized on the different technical choices, by showing the framework's flexibility through the new functionalities recently introduced, such as: Transactions, Diagnostics, Monitoring, Management of LHC Critical Settings, communication with PLC devices and Composition. To conclude we will present the extensions foreseen in the short term.

FESA

Application Domain

The Application Domain, for which FESA has been developed, is the real-time software running on front-end computers at CERN. This software, called equipment software, provides access to the accelerators' physical elements that interact with the beam, by handling operators' request (property interface) and dealing directly with the hardware.

What is FESA?

The Front-End Software Architecture, known as FESA, is a complete environment for the equipment specialists to design, develop, deploy and test their equipment software, called a FESA class. The primary reason to develop such an infrastructure is to standardize, speed-up and simplify the task of developing front-end software. The FESA infrastructure includes the following components:

- Object-oriented Real-Time Framework: a fundamental technical choice. Instead of providing individual re-usable components that implement common functionalities needed by almost all applications (toolkit and libraries), we decided to focus on the structure and the flow control of our application domain, and to capture the outcome of this analysis into a framework. The goal is to spare the programmer from having to define the architecture of each new application, because the framework is the application for the equipment real-time software domain. The framework orchestrates the activity, and when necessary calls the routines provided by the application developer to provide application-specific behaviour.
- Graphical Tools: developing a FESA class requires the developer to produce three XML documents: Design, Deployment and Instantiation. In order to supply them, the developer uses a graphical application, which is basically a generic XML editor, configured by dedicated W3C XML Schemas [1]. These respective XML Schemas are the following:
  - FESA Design Schema: drives the Design tool. It encodes the meta-model, i.e. the model of all possible models of equipment software. The FESA meta-model forces the equipment specialist to think of an equipment design as a set of conceptual objects:
    - A public Interface, as a collection of get/set/subscribe properties. Equipment is controlled through remote invocation of these properties across the Control system Middleware (CMW [2])
    - A device-model, representing the software abstraction of an underlying hardware device, composed by a set of fields (data-holder).
    - A set of server actions, implementing the properties get/set services.
    - A set of real-time actions transferring data to/from the hardware device from/to its software device-instance representation.
    - A set of logical events which trigger the equipment's real-time activity.
    - A set of triggering rules, binding logical events and real-time actions.
  - FESA Deployment Schema: drives the deployment tool used to deploy a FESA class on a particular front-end.
  - FESA Instantiation Schema: drives the Instantiation tool, used to configure a set of device instances on a front-end on which the FESA class is deployed.
- Code generation: the specification of the design in XML, allows us to automatically generate the appropriate source code in order to specialize the framework according to the equipment specific needs. For this purpose we have used other XML-based technologies such as XSL (eXtensible Stylesheet Language) [3]. XSL is a language for creating style sheets that specify how to transform elements in XML documents into C++ source files. During the "Fesa Synchronize" phase, a set of XSL files is applied to the design XML document, in order to populate the generated code package and instantiate the skeletons of the custom actions and Makefiles.
- Test environment: again, relying on the use of XSL, FESA provides an automatically generated Java GUI that allows access to every defined property of any declared device instance.

The successive steps of a FESA development process are shown in Figure 1.

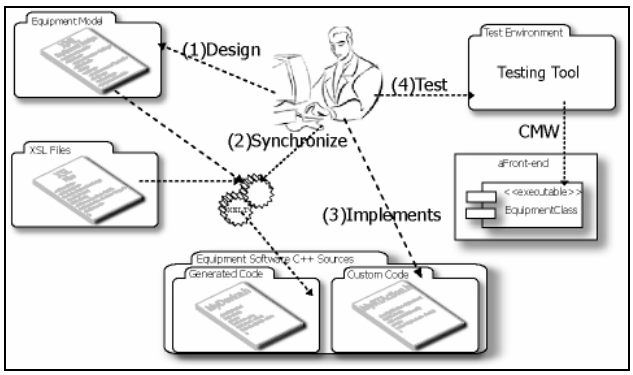

Figure 1: successive steps to develop equipment software using FESA (deployment and instantiation are not shown).

Equipment Software Reusability

CERN is composed of a chain of accelerators, each of them having dedicated timing domains that synchronize all the physical elements according to the characteristics of the beam circulating in the accelerator. So, even if the accelerators are composed of the same physical elements (Magnets, Beam Instrumentation, RF cavities, etc…), controlled through the same layer of hardware, the dynamic behaviour of those elements change from one accelerator to another. To ensure equipment software portability across CERN accelerators, FESA provides a complete abstraction of the timing system at several levels:

- Design: designing equipment software in FESA, involves defining the scheduling as a set of scheduling-units, each defining two rules:
  - Triggering-rule: specifies the binding between an event and a real-time action. The event specified in the design, is a logical event (abstract event), that will be concretized into a real accelerator timing event. Each time the equipment software is deployed, a set of device instances are configured with their accelerator specific events.
  - Device-selector criteria rule: specifies the grouping criteria to build homogeneous device collections managed by the real-time action. In terms of timing, this criterion is: "devices that need to be operated at the same time in the accelerator cycle". To this end, FESA defines a device-interrupt field, in order for the device instance to specify the timing it is connected to. In this way, a single triggering rule binding a logical event to a real-time action, can produce as many instances of this real-time action as there are homogeneous groups of devices instances, resulting from the device-selector.
- Custom code implementation: for each real-time action class, the equipment-specialist has to implement (in C++), the body of the "execute(RTEvent*)" method. The input parameter of this method refers to an RTEvent object that contains the 'wake-up' context by which the action was triggered. This context, managed transparently by the framework, contains timing information related to the accelerator, and is required as a parameter when the custom code invokes the device field's get and set, in order to handle the multiplexing of settings. (Multiplexing: accelerators infinitively play a programmable sequence of successive cycles, called USERs or "Virtual Accelerators", based on a basic period (1.2s), and orchestrated by the timing system, in order to provide beams for the different destinations. Switching from one cycle to the next, causes a switch of equipment setting from one USER to the next).

STATUS REPORT

Are the Results the Expected Ones?

In spite of the huge diversity of devices, such as Beam-Loss monitors, Kickers, Cryogenic systems, Pick-ups, etc…, FESA has successfully standardized a high level language and an object oriented framework to describe and develop portable (meaning across CERN's accelerators) equipment software. FESA reduces the time spent developing and maintaining equipment software and brings a strong consistency across all equipment software deployed over all accelerators at CERN. The high level language, used to design equipment software, makes it really easy to understand the structure and the dynamic behaviour of any equipment software, just by examining the design-document. The FESA development environment provides a tool to model equipment software, and keeping this model and implementation synchronized.

Managing Extensions, Evolutions

As the cornerstone of the CERN Control system, FESA must accommodate evolving requirements. These last two years, we have introduced important new features and this chapter reports the methodology we have put in place to make those evolutions straightforward.

- FESA versioning policy: FESA evolution is mainly driven by requirements coming from Equipment Groups. Since those partners have their own time constraints, it's not possible to force them to follow a unique FESA release. Hence, we have decided to maintain three operational releases, which is a good compromise to force equipment specialist to migrate to the new version, granting some flexibility in terms of planning.
- Retrofitting: Each FESA release comes with the appropriate tool, to retrofit any equipment software from one FESA version to the new one. This tool, will automatically patch the design document as well as developer's code. These last two years, we have introduced important changes at all levels (meta-model and Framework), and we have demonstrated that the retrofit process is not time consuming nor a source of pain for the equipment specialist.

Main Features Recently Introduced

- FESA class Composition: two types of composition relationships between FESA classes have been introduced:
  - "Has a": from FESA class A, you want to have a reference to a device-instance managed by FESA class B. You can access it using class B's interface (properties).
  - "Implements interface": for complex accelerator devices, such as RF cavities, equipment specialists want to decompose them into sub-systems, each providing a dedicated interface in order to give a specialized view of the system for different categories of clients.
- Timing Simulation: is intended to help the development and testing of equipment software without hardware timing. From the instantiation tool, the equipment specialist can simulate any accelerator timing scheme, in terms of sequence of cycles, and for each cycle, any sequence of events.
- Run-time diagnostic: the first logging system we provided was based on levels, borrowed from log4j. This approach, where debugging messages are filtered according to the log level trace, has proved lacking in flexibility. We have decided to move to a "topic-oriented" diagnostic, in order to have a finer granularity in the trace options. The framework defines a series of topics such as "Notification", "EventTracking", "RTActionProfiling," or "ServerActionProfiling"... in order to diagnose the complete control flow of the equipment software. In addition, the system allows the equipment specialist to define their own topics, to diagnose precisely the custom part of his equipment software.
- Monitoring: the purpose of monitoring is to permanently survey the control flow of any equipment software in a non intrusive way, and raise an alarm if something is not working properly. In the context of the FESA framework, it is straightforward to control the real-time activity by adding some reference points in strategic places, in order to detect for instance, accumulation of real-time events showing clearly that the custom-code is not able to treat the triggering events.
- Programmable Logic Controller integration: more and more accelerator devices are connected to PLC's. These industrial systems are connected to a standard Ethernet network, and have the ability to communicate with the Front-end using TCP/IP. As PLC programmers have no desire to deal with Linux, or C++ programming, the integration has to be completely automated. To this end, we have defined a branch ("plc-class") in the FESA meta-model, as a restriction of a standard class. Using the standard FESA design tool, the PLC specialist defines the data-structure that will be exchanged between the PLC and the front-end, in terms of Acquisitions, Settings, and Configuration. This design is used to automatically generate the FESA plc-class (no coding is required), and the data structure to be downloaded into the PLC.
- Critical Settings Management: with the arrival of LHC, it has been recognized that some device's settings are related to critical parameters of the system, and that the given value should be validated before actually setting some property. The technical solution chosen is based on public-key cryptography and a digital signature. To this end, we have extended the meta-model to allow the equipment specialist to flag, in the design, the properties which are critical. Setting a property marked as critical, will require a signature that the client has to provide with the settings data. This new service is managed transparently by the framework.
- Transaction: this facility guarantees that several settings acting on different devices deployed on various front-ends will be all taken in account at the same time, or none of them in case of an error. FESA transaction is a two phase commit transaction. The transaction coordinator sends settings to all the devices involved in the transaction, and waits until it has a reply from each of them. If all the devices send an agreement message, the coordinator asks the timing system to send a "CommitEvent". If at least one device failed, an AbortEvent is sent instead.

All these recent extensions have proven the flexibility, and the capability of the complete FESA infrastructure to handle evolution.

Short Term Extensions

For the coming short term (next year), we would like to investigate some new extensions:

- Hardware integration.
- Composition, versus inheritance, between FESA classes.
- FESA IDE: building a complete integrated development environment using the Eclipse platform.

REFERENCES

[1] W3C Schema specification http://www.w3.org/XML/Schema
[2] "Remote Device Access in the New CERN Accelerator Controls Middleware", ICALEPCS'01, San Jose, California, USA, p. 496.
[3] W3C XSL specification http://www.w3.org/Style/XSL
[4] Equipment Software Modeling for Accelerator Controls, ICALEPCS 2005
[5] Use of XML technologies for data-driven accelerator controls, ICALEPCS 2005