CERN FRONT-END SOFTWARE ARCHITECTURE FOR ACCELERATOR CONTROLS

A. Guerrero, J-J Gras, J-L Nougaret, M. Ludwig, M. Arruat, S. Jackson,
CERN, Geneva, Switzerland

Abstract

To overcome the current diversity in AB [1] front end equipment software and pave the way towards LHC [2] for efficient development, diagnostic and maintenance in this area, the CERN Accelerator Controls group launched in April 2003 a project to develop the new CERN [3] accelerator standard infrastructure for front end software. This development is based on the infrastructure recently born to handle the SPS beam measurement systems and extends it to handle the PS and SPS multi-cycling schemes, the future requirements needed for LHC as well as providing a good backward compatibility with the existing infrastructures. The project, approach and first deliverables are presented.

INTRODUCTION

The ***F**ront-**E**nd **S**oftware **A**rchitecture* framework, known as FESA [4], is a complete environment for the equipment specialists to design, develop, test and deploy real-time control software for front-end computers. This framework will be used from now on to develop the LHC rings and injection chain front end equipment software.

Based on the BISCoTO [5] tools and functionality, the primary objective of this framework is to standardize, simplify and optimize the task of writing front end software. It encompasses the following:

- A *generic model* that captures the recurrent aspects of real-time software programming in the domain of accelerator control.
- A *method* whereby equipment–specialists reuse / apply the generic architecture and tailor it to their specific needs on a case-by-case basis.
- A set of *software tools* to support equipment-specialists, users and the exploitation team at all stages of the development and during operation.

ARCHITECTURE MODEL

In order to ensure good productivity and maintainability of the code, a clear separation between the generic and the equipment specific code has been enforced, as illustrated by Figure 1. This class diagram shows:

- The generic layer consisting of a set of inter-dependant base classes (represented as fine-contoured rectangles). Together, these classes form the backbone of reusable front-end software.
- The set of application-specific concrete classes (represented at the bottom of the diagram with bold contours) are derived by the equipment-specialist from the framework base classes.

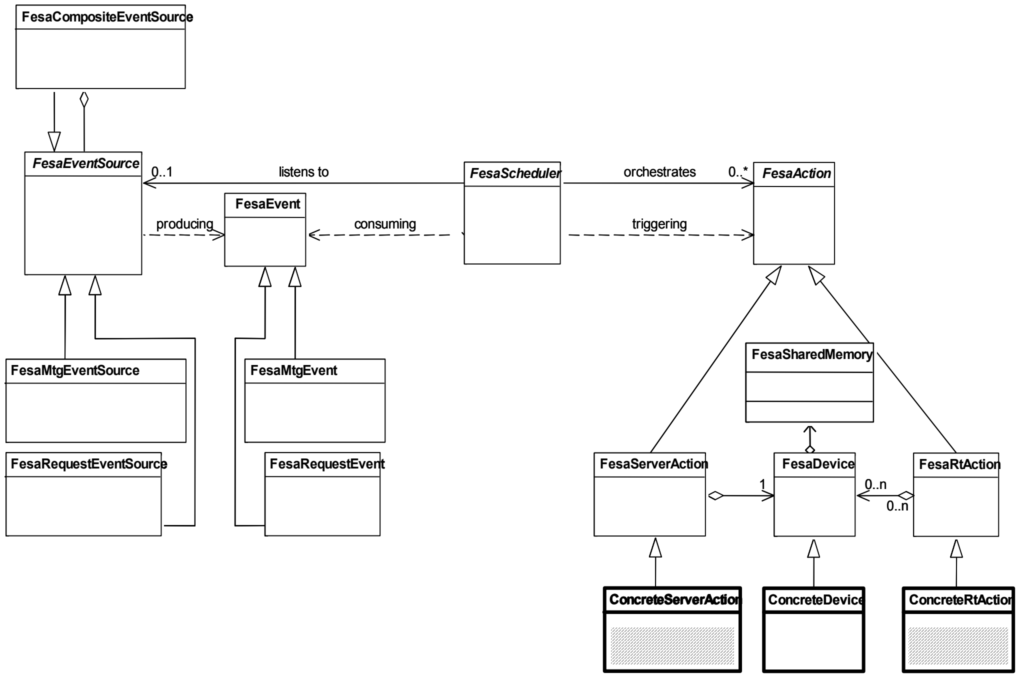

Figure 1: Class diagram representing the static view of the FESA framework - Only classes filled with a dashed-pattern actually require custom-coding from the front-end software developer

Automatic code generation forms a significant part of the concrete class layer. The ConcreteDevice class, which represents the software abstraction of the equipment, has its interface (i.e. list of properties) fully described in the configuration database. The information is then used by the framework to automatically generate corresponding C/C++ code (the front end ConcreteDevice class) as well as Java code (for the generic graphical test application).

As a result, the ratio between the generic/generated code and the developer specific code in the front end usually equates to around 10 to 1 with most of the added code being limited to equipment specific actions.

Structure

As shown in Figure 1, the framework's base layer is composed of the following classes:

- At the heart of real-time activity, the *FesaEventSource* acts as a pacemaker by firing *FesaEvent* instances. A *FesaEvent* wraps hardware interrupts, machine events or client requests as objects. The *FesaCompositeEventSource* class acts as an event-source concentrator by providing a common channel through which different kinds of events, fired by different event sources running in their own threads, may flow in. This plug-in event source approach allowed us to incorporate different machine timing libraries as well as the middleware of the PS and SPS machines whilst hiding these differences from the equipment software developer. This approach also allows event source simulation.
- The *FesaAction* is the basic work-unit carried-out by the equipment software, and is where the equipment specific functionality is coded. The *FesaAction* class is sub-classed by the *FesaRtAction* and *FesaServerAction* classes which handle hardware access and client requests respectively. A *FesaRtAction* typically executes code that deals with the hardware, is triggered by an interrupt, and is subject to tight timing constraints. A *FesaServerAction* typically fulfils a request from an operator in the control room and runs at a lower priority. For each of these actions, the developer supplies an `execute()` method in which he/she has direct access to the targeted device (via the `ConcreteDevice:device->getPropertyName()` method).
- The *FesaDevice* class is a data-holder that reflects the state of an underlying hardware device. *FesaDevice* attributes fall into several categories including but not restricted to the following: settings, acquisitions and state-variables.
- The *FesaScheduler* singleton object listens to the event source and following an event, it triggers an appropriate action according to predefined logic (usually a simple associative map which defines a sequence of elementary actions to be executed for each type of event. See Table 1).

Behaviour

The scenario diagram of Figure 2 depicts the typical behaviour of an RT-task observed after customizing the FESA framework for a specific device and set of actions.

Firstly, the RT-task subscribes to events fired by the central timing process (or any other HW sources). This process orchestrates the accelerator's pulsed-mode of operation, and the RT-task block-waits on event reception. Whenever an event occurs, the framework examines its type and triggers the appropriate action as specified in the equipment configuration (Fig 2).

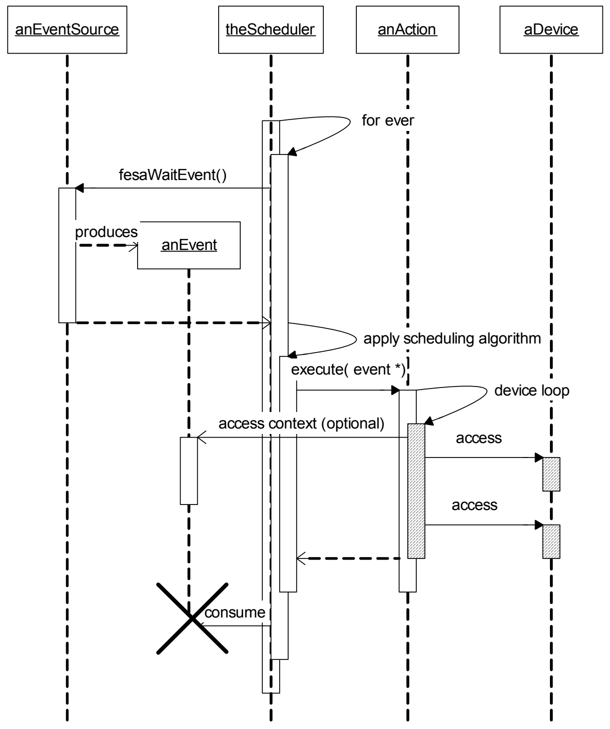

Figure 2: Sequence diagram of real-time handling

In the sequence diagram, activities represented by rectangles filled with a dashed pattern, are those that differ from one RT-task to another and that the programmer does code. This diagram illustrates two key points:

- The specific (i.e. custom) part of an RT-task essentially lies in the body of the elementary actions that the RT-task triggers.
- The scheduling algorithm whereby the dispatcher triggers appropriate actions in response to incoming events is provided by the FESA framework. Furthermore, the scheduling algorithm is fully configurable by the equipment specialist. The equipment specialist merely fills in a table that links the triggering of a particular action to the occurrence of a particular event for a predefined subset of devices (see Table 1). This table is retrieved at the initialization of the front-end software, which then updates the associative map it relies on for scheduling.

Table 1: Configuration of the event-action map that defines the triggering of real time actions upon event occurrences. A machine Beam In event will launch the Acquire action on every device of device type 'Proto'

Source name	Event name	Action name	Action Arg	Device Selector
-	Init	Init		All devices
MTG	Warning Beam In	Prepare		All devices
MTG	Beam In	Acquire		DeviceType = Proto
UserCmd	Emergency	Stop	Emergency	All devices
UserCmd	Stop	Stop	Normal	All devices

WORKFLOW & SOFTWARE TOOLS

In order to develop equipment software within FESA the framework must be customised to suit the equipment specific needs. To assist in this process, a set of tools is provided to support the equipment specialist at all phases of development.

The design phase

Firstly, the developer must describe his/her new equipment interface, internal structures and real time behaviour. The FESA configuration tool allows both the configuration of internal structures as well as the population of the underlying data. As the front-end uses code generated by this tool, it knows where to find and load the data on each reboot.

The coding phase

Based on the equipment description, the configuration tool also generates the related C/C++ code which handles both the shared memory population and the external client access. Java code is also generated and is used for client access via the generic graphical tool, the Navigator, which allows a developer to drill-down to any property defined in the equipment.

The configuration tool also deploys code which the developer will have to modify, and in which he/she will have to implement the equipment specific functionality, i.e. the declared RT and Server actions. Even without any developer input, the equipment server already compiles. The resulting communication process provides Get and Set access to every declared device property and the 'empty' server action via the Navigator. The real time process can launch the 'empty' RT actions in accordance with the declared event-action map. The specific actions can then be coded by the developer.

The testing phase

As mentioned earlier, the Navigator is a generic application, driven by the automatically generated Java stubs which allow access to all properties in all declared device instances of the equipment class. It supports access to compound data types as well as atomic command response.

The framework also provides system-level services to monitor the processes and follows their activity without degrading the real time performance of the front end processes. A front-end process activity tool allows the remote surveillance of any FESA process, and allows the run-time control of parameters like the verbosity level and the logging history depth.

The framework also incorporates a hardware driver generation tool, developed for the BISCoTO [5] project. This tool allows the automatic generation of a VME board driver with the corresponding access library based on its registers and address mapping description. It also systematically generates a simulation driver, which emulates the same interface based on kernel-space memory instead of real VME access. This feature, when combined with the event source simulation, allows the developer to complete his/her equipment software even if the hardware or timing events only exist on paper.

PROJECT STATUS

The architecture described here is currently being designed and implemented by a joint-team of both end-users and maintenance engineers. It is now in the functioning prototype phase.

The FESA framework will be made available on the AB operational front end platforms, i.e. Linux and LynxOS real time OS. It has been developed in C++ but will be available to pure C developers via a C interface in the near future.

The software tools (configuration, navigator and process activity survey tools) are written in Java and are thus platform independent. They are currently used on both the Linux and Windows platforms.

The first version of the framework will be delivered to the equipment group developers in November 2003 and will be deployed operationally on several front-ends in the LHC injector chain in 2004.

REFERENCES

[1] http://public.web.cern.ch/public/
[2] http://ab-div.web.cern.ch/ab-div/
[3] http://user.web.cern.ch/user/Index/LHC.html
[4] http://sl-div-bi-sw.web.cern.ch/sl-div-bi-sw/Activities/FEComSA/entry.htm
[5] A. Guerrero and S. Jackson: "Common Templates and Organisation for CERN Beam Instrumentation Front-End Software Upgrade" (these proceedings).