FESA3
THE NEW FRONT-END SOFTWARE FRAMEWORK AT CERN AND THE FAIR FACILITY

Alexander Schwinn, Solveigh Matthies, Dorothea Pfeiffer(GSI, Darmstadt)
Michel Arruat, Leandro Fernandez, Frank Locci, David Gomez Saavedra (CERN, Geneva)

Abstract

Currently the LHC (Large Hadron Collider, located at CERN/Switzerland) is controlled by the use of FESA2.10 (FrontEnd Software Architecture v. 2.10) classes. FESA3 is not only an update of FESA2.10, but a completely new approach. GSI plans to use the FESA system at the complex FAIR facility.

One of the main reasons to introduce FESA3 was to provide a framework which can be shared between different labs. This is accomplished by splitting up the FWK into a common part, which is used by all labs, and a lab-specific part, which allows e.g. a lab dependent implementation of the timing-system.

FESA3 is written in C++, runs a narrow interface (Remote Device Access, a middleware which encapsulates CORBA), supports multiplexing of different accelerator-cycles, is completely event driven and uses thread priorities for scheduling. It provides all FESA2.10 functionalities and additionally introduces several new features.

FESA3 is integrated in the Eclipse IDE as a plugin. Using this plugin, the user can easily create his FESA-class design (xml file), generate the C++ source code, fill the device-specific methods, and deploy the binary on a front end.

As well as the framework the Eclipse plugin has a lab specific implementation.

An operational release for FESA3 is planned end of 2010.

THE PURPOSE OF FESA3

FESA3 is a software framework which provides an easy way for developers to produce device classes by generating most of the code automatically. It supports multiplexing of different accelerator-cycles and many other features which can be used by the class-developer. The main purpose of the framework is to provide an common and unified way to develop device classes. This approach saves a lot of work and simplifies debugging, documentation and code adoption for the class-developer and all involved parties.

THE ROOTS OF THE FESA FRAMEWORK

All early versions of the FESA framework were developed solely by the CERN facility. FESA3 is the first release which is developed as an collaboration between CERN and the GSI. This collaboration was the main reason to restructure some of the Fesa2.10 fundamental internal parts and to finally go for a new major release.

FESA3 continues to provide all services from older versions and as well extends the common approach by additional services which where demanded by the CERN user community.

FESA3 AT THE FAIR FACILITY

For the FAIR facility several new accelerator installations will be built at GSI.

Central aspect is an increased number of research programs resulting in up to five beams in parallel. The FAIR facility will be controlled by a new control system which will be able to support all aspects of the complex GSI/FAIR operations on a common technical basis. The control system for the FAIR facility currently is in the design phase.

One part of this new control system will be the device software which runs on the front ends. FESA was choosen as software framework since it already proved itself at the LHC at the CERN facility and allows to pass the device-specific implementation directly to the device expert.

CLASS DEVELOPMENT WORKFLOW

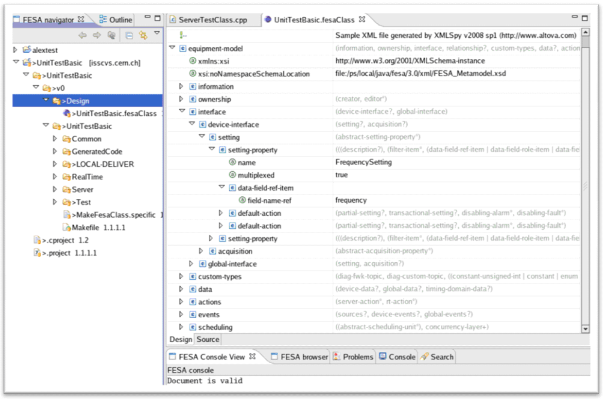
The FESA3 Eclipse-plugin guides the class developer on his way to develop a FESA3 class. The following steps have to be performed to do so:

1. Design

   In the first step the developer needs to design his class according to his needs. This process involves the specification of *Properties*, *Fields*, *Server-* and *RealTime-Actions* and their dependencies on each other. The design itself is done via a comfortable XML editor, which is integrated in the FESA3 Eclipse-plugin and coupled to an XSD schema for validation.
   (see figure 1)

2. Code Generation

   Code generation may be started in the plugin if the class design is valid. An XSLT engine generates C++ code using the class design as input.



```
!--                                   Sample XML file generated by XMLSpy v2008 sp1 (http://www.altova.com)
equipment-model                       (information, ownership, interface, relationship?, custom-types, data?, action
  xmlns:xsi                           http://www.w3.org/2001/XMLSchema-instance
  xsi:noNamespaceSchemaLocation       file:/ps/local/java/fesa/3.0/xml/FESA_Metamodel.xsd
  information
  ownership                           (creator, editor*)
  interface                           (device-interface?, global-interface)
    device-interface                  (setting?, acquisition?)
      setting                         (abstract-setting-property*)
        setting-property              (((description?), (filter-item*, (data-field-ref-item | data-field-role-item | data-fie
          name                        FrequencySetting
          multiplexed                 true
          data-field-ref-item
            field-name-ref            frequency
          default-action              (partial-setting?, transactional-setting?, disabling-alarm*, disabling-fault*)
          default-action              (partial-setting?, transactional-setting?, disabling-alarm*, disabling-fault*)
        setting-property              (((description?), (filter-item*, (data-field-ref-item | data-field-role-item | data-fie
      acquisition                     (abstract-acquisition-property*)
    global-interface                  (setting, acquisition?)
  custom-types                        (diag-fwk-topic, diag-custom-topic, ((constant-unsigned-int | constant | enum
  data                                (device-data?, global-data?, timing-domain-data?)
  actions                             (server-action*, rt-action*)
  events                              (sources?, device-events?, global-events?)
  scheduling                          ((abstract-scheduling-unit*), concurrency-layer+)
```

```
FESA console
Document is valid
```

Figure 1: screenshot of the class-design in the FESA Eclipse-plugin

3. Implementation

   The code generation provides methods which need to be completed by the class developer. Those *Server- and RealTime-Actions* allow to use specific device drivers for the hardware. External driver libraries can be included in a class specific make-file.

4. Instantiation

   Since the FESA3 class may run on different front ends, the developer needs to create an instantiation document per frontend. Inside this document all configuration parameters for this specific frontend are stored. Similar to the class-design, the instantiation document is generated by the plugin and can be edited within it's xml editor.

5. Compilation

   As soon as the implementation is finished, the compilation and linking process may be started. The outcome will be a class binary for a predefined platform, which is ready to run.

6. Deployment

   The resulting binary, the instantiation file and all other dependent files need to be placed on the frontend for which they were configured and to which the device-hardware is connected.

7. Execution

   Finally the class can be executed and debugged. The FESA3 navigator-tool may help to build up a connection to the class and debug all possible scenarios.

The described workflow is not strictly forward but also allows to roll back and redo any step which is necesarry.

FUNCTIONAL OVERVIEW

Basic Internal Design

As shown by the use-case diagram below, request-handling and hardware control are the two complementary services equipment-software has to model. The two differ very much in nature since request-handling is an on-demand service, whereas hardware control is subject to tight real-time constraints. Obviously, request handling must run at a lower level of priority and shall not be able to preempt the real-time task. In order to decouple the two, equipment-software includes a software abstraction of the device. Thanks to this abstraction, an operator does not directly see the hardware device, but rather accesses it through the so called *Server side*. (see reference [1])

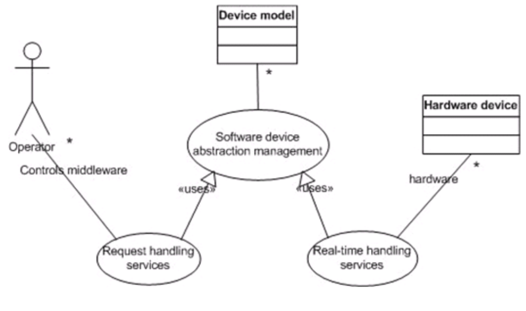

Device model
*
Operator
*
Controls middleware
Software device abstraction management
«uses»
«uses»
Hardware device
hardware
*
Request handling services
Real-time handling services

Figure 2: Separation of *Server and RealTime side*. [1]

The system is split in two logical layers: *RealTime*, which implements all parts that are directly triggered by events and *Server* which models the equipment interface and implements the middleware access. Both services are physically implemented on the same hardware platform. It is possible to run these two services in the same process, or in two separate processes. Devices are implemented as objects in the object-oriented software terminology. Each FESA3 class represents a devicetype and allows to manage different instances of this devicetype. A FESA3 equipment can represent a collection of different FESA classes, which depend on each other.

The event driven RT-system

In FESA3 *RealTime-Actions* can be triggered by different types of event, from various event sources.(see figure 3) The possible source types are listed here:

- Timing Event

  This event source is meant to be the real accelerator timing. Different events corresponding to different cycles and machines are received with this source if the frontend is connected to the timing receiver hardware.

- Timer Event

  A timer event is launched by a internal clock on a periodic interval. The developer can configure this interval in the instantiation document per frontend.

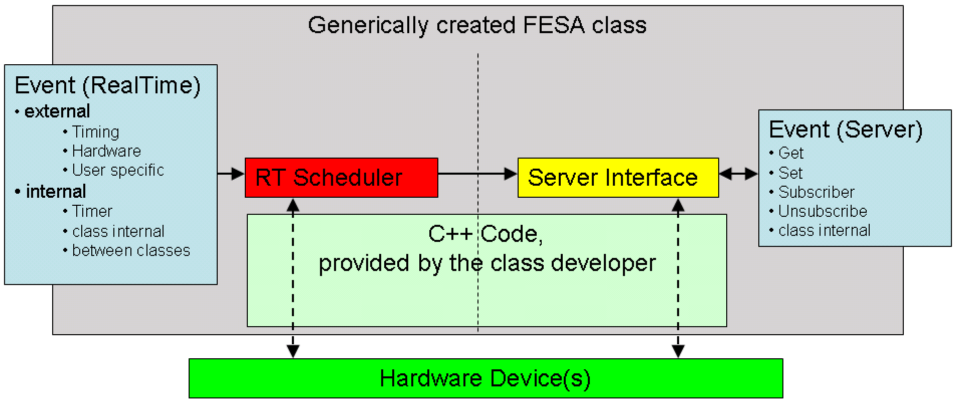


Figure 3: Basic event based functional structure of the FESA3 framework

- Custom Event

  In FESA3 the developer as well has the possibility to implement his own customized event source. This event source is used to support all other hardware and software event sources which do not use the accelerator timing.

- On Demand Event

  To communicate between *Server* and *RealTime* part of a FESA3 class there is not only the notification queue, which connects the *RealTime* to the *Server part*, but also the *On-Demand* mechanism, which works the other way around. Via socket connections it allows to trigger *RealTime-Actions* from the *Server side*.

- On Subscription Event

  FESA3 allows to establish dependencies between different FESA3 classes. E.g. one FESA3 class can subscribe to properties of another FESA3 class by using this event source.

The client interface

The client has different possibilities, to communicate with a FESA3 class using the RDA client interface. Access methods for the client are *Get*, *Set*, *MonitorOn* and *MonitorOff*. The API to these methods is a narrow one. Figure 4 shows the relation between the RDA *DeviceServer*, the middleware layer and the client.

To specify the proper attribute(s), there are several parameters:

- Property

  The property describes a collection of data, which can be obtained or modified with different client access methods.

- Device

  A FESA3 class represents a device type. A single FESA class can control many devices of the same type. Within the parameter *Device* the client can specify the proper device instance which is to access.

- CycleSelector

  On a multiplexed property, with the cycle selector string the client can select the cycle (virtual accelerator), he wants to work with. On a property which is not multiplexed the cycle selector can stay empty.

- Value

  Get methods retrieve data as an instance of the type *rdaData*. *RdaData* can store an array of mixed data types. Besides the data itself, each entry allows to store additional information.

- Context

  The context is used to pass parameters (filters) to the properties of a FESA3 class.

- ReplyHandler

  Monitor on calls require the implementation of reply handlers. As soon as new data arrives, the middleware triggers the reply handler in which the data is processed.

- Request

  This class is the virtual handle to a subscription. It is filled by the *MonitorOn* call and is used to keep track on a subscription and to terminate it if it is not needed any more.

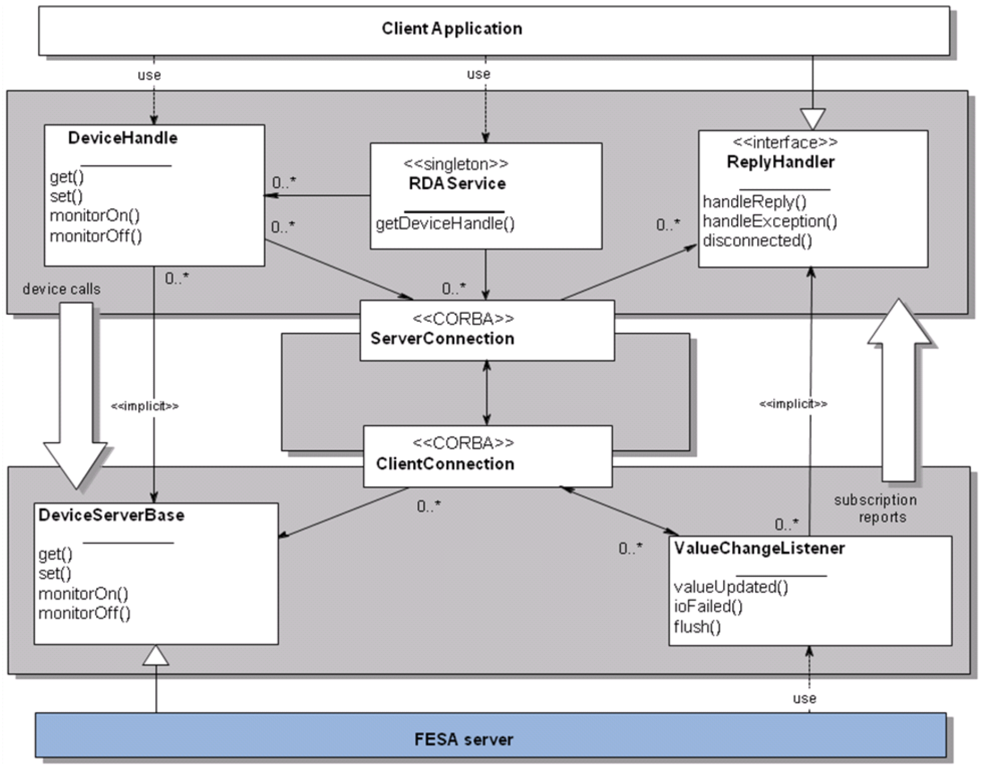


Figure 4: The CERN RDA - middleware layer. (see reference [2])

class relationship

Unlike FESA2.10, FESA3 provides three different ways of class relationship, association, composition and inheritance:

- Association (see figure 5)

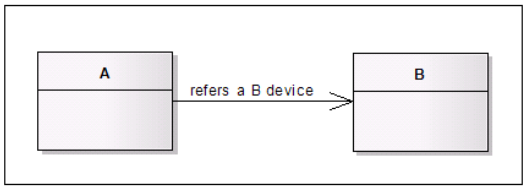


Figure 5: Association between two unique FESA3 classes

  An association specifies a light coupling between two FESA3 classes. The two classes run independently and do not rely on each other (e.g. class *A* may shut down while class *B* is still running). The two classes can be deployed on different frontends.

- Composition (see figure 6)

  Using a composition the developer can create a strong coupling between FESA3 classes. The deployment of class *A* means to deploy the whole class-tree. As well it is possible to start *B* with only *C* and *D* as sub-classes and without *A* as a smaller composition. The lifetime of the composition depends on the lifetime of the base-class. All classes need to run on the same frontend. All classes within the composition are standard FESA3 classes and can be used seperately as well.

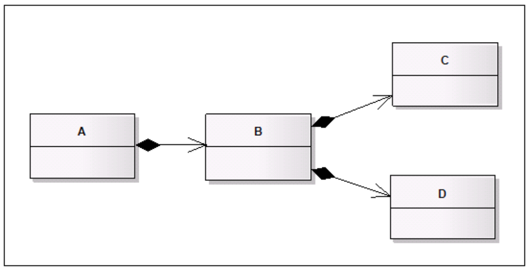


Figure 6: Composition of many FESA3 classes, represented as one class

- Inheritance (see figure 7)

  The definition of inheritance in FESA3 consists of tree characteristics:

  – *Properties/RTActions* defined by a baseclass are available for any subclass.

  – *Properties/RTActions* defined by a baseclass can be overridden (explicitly).

  – The device model of a derived class fully inherits from the device model of its baseclass.

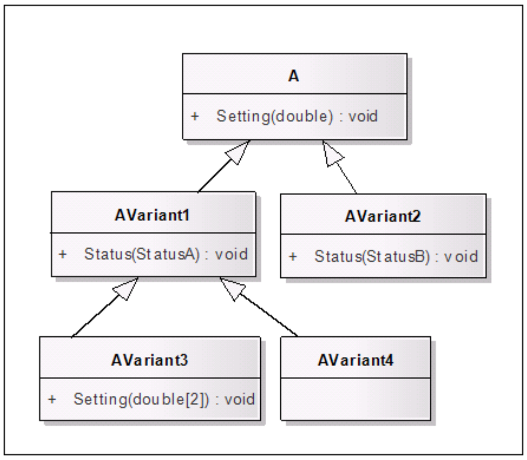


Figure 7: Inheritance between different FESA3 classes

The split between framework and lab part

The FESA2.10 framework was strongly coupled with the CERN Oracle database, the CERN Timing and several file paths. As one of the major changes this strong coupling has been removed in FESA3. Each institute which is using FESA3 now has to provide a sub-package where institute-specific code can be used. This split does not only touch the C++ code. The whole process of creating a FESA3 class is involved. Now it is possible to have a specific metamodel which triggers an adapted code generation. As well the FESA3 Eclipse plugin can be adjusted to the needs of the particular institute.

OUTLOOK

Currently FESA3 still is in the pre-beta phase. An operational beta for FESA3 will be released end of 2010. For later releases the following features and tasks are planned:

- Transaction

  On larger accelerators the possibility to synchroniously trigger a *Set* for many frontends is needed. This service is called ”transaction”.

- On-change/deadband support for subsribers

  This service allows clients to choose, if they will get notified if a value did not change at all, or if it changed only within a predefined deadband.

- Tests and benchmarks of the framework

  Tests of the FESA3 performance in terms of reaction-speed, data throughput and CPU usage need to be done.

REFERENCES

[1] A. Schwinn, D. Pfeiffer, R. Baer, “GSI-FAIR Baseline Technical Report - Front-End Software Architecture”, GSI, Germany.

[2] Kris Kostro, Joel Lauener, Nikolai Trofimov, Wojciech Gajewski, Ilya Yastrebov, “http://cmw.web.cern.ch”, CERN, Geneva