



*Document Title*

**FAIR Control System Development Guideline  
“Software Architecture Guideline”**

*Document Name*

**F-DG-C-03e**

*Date yyyy-mm-dd*

**2012-08-31**

## **Abstract**

This document describes and explains the principles of the Software Architecture for the FAIR accelerator control system. It documents a common base of architectural aspects for the FAIR accelerator's control system software and references other Guidelines or Detailed Specifications of components to use.

## Table of Contents

1.	Purpose and Classification of the Document .....	3
1.1.	Responsibilities .....	3
2.	Scope .....	3
3.	Introduction.....	4
4.	Guidelines .....	4
4.1.	System Modularization and Module Interfacing.....	4
4.1.1.	Service Orientation.....	4
4.1.2.	Interfacing .....	5
4.1.3.	Usage .....	5
4.1.4.	API implementation .....	5
4.2.	Data Persistence .....	5
4.3.	Access Rights .....	5
4.4.	Error Handling.....	6
4.5.	Logging.....	6
4.6.	Software Configuration .....	6
4.7.	Technology Constraints .....	6
4.8.	IT Infrastructure Constraints.....	6
I.	Related Documentation .....	7
II.	Related Documentation .....	7
III.	Document Information .....	7
III.1.	Document History .....	7

## 1. Purpose and Classification of the Document

The purpose of this document is to specify the development guideline as a directive for developing control system software for FAIR. Adherence to this guideline will improve product quality and maintainability of the FAIR accelerator control system.

The development guidelines complement the technical guidelines and detailed specifications for the FAIR control system in providing general rules and regulations for control system development.

Whenever regulations and requirements are specified in the General Specifications, Technical Guidelines, Common Specifications or Detailed Specifications of the Control System they are only referenced in this document. The related documents are listed in Appendix II.

No legal or contractual conditions are treated in this document. All related information is given in the General Specifications for FAIR II.

### 1.1. Responsibilities

The responsibilities with respect to changes and modifications of the present document are entirely in the hands of the Controls Department of the GSI Helmholtz Centre for Heavy Ion Research GmbH (GSI) Darmstadt.

For initial information please contact the administration of the Controls Department.

Further information on the organigram, names of responsible persons and task leaders, as well as the agreed document release and approval procedure is summarized in the organizational note 'Controls Project for FAIR'.

## 2. Scope

The Software Architecture Guideline holds for all components of the FAIR accelerator control system's application and service tier. For Frontend Controller Software aspects and accessing Frontend Controller Software see [2].

The design and implementation of the software components of the application and service tier must adhere to the principles, constraints, and technologies as specified in this document.

This document does not cover software development methodology [1]. This document does not cover user interface guidelines [11]. This document does not cover functional aspects of any application or service component [9]. This document does not cover device-modeling or equipment-software's functional aspects [12].

### 3. Introduction

Most likely, within the foreseen lifetime of 30 years of the FAIR facility the computer hardware and software technologies will drastically change. Thus, the Software Architecture must ensure a high level of maintainability and exchangeability of components.

For the whole life cycle of the accelerator complex it is assumed that the software components of the application and service tier can be organized in a modular and distributed way. We do not assume that the technology, i.e. operating system, programming language, and development frameworks will remain constant. Therefore, to enable flexibility in maintaining, developing, and replacing components of the application and service tier we refer to and adapt the concept of service oriented architectures wherever possible.

Another important aspect of maintainability is to enforce a uniform usage of core services like data-persistence technologies, organization of access rights, localization, and definition of error handling.

Also, we define the currently expected constraints given by the IT infrastructure technology.

## 4. Guidelines

### 4.1. System Modularization and Module Interfacing

To get an overview about functional aspects of the accelerator's control system software stack see [1]. Here we describe non-functional and technical aspects of modularization.

Monolithic architectures, e.g. Java-EE or .NET, implement modularization and module connectivity within their own frameworks. Notwithstanding the well-known benefits of such architectures, we had to consider the domain specific life cycle of the accelerator's control system of more than 30 years and its impact on the feasibility of such architectures. In fact, it showed that a Service Oriented Architecture (SOA) fits better the requirement of long term maintainability.

#### 4.1.1. Service Orientation

In the context of SOA a service is a module (software process) which connects at run-time to other modules and which is otherwise independent from other modules. The connectivity must not depend on the chosen technology of any module. Such modules might be components of the service-tier or application-tier. In many cases it is even helpful to separate user applications in modules with a GUI component and the business logic. In a service oriented architecture such a module is for itself not more maintainable than a module in a monolithic approach. But since existing modules can be replaced or new modules can be added, and be implemented by means of new technologies, the maintainability of the whole software system increases. Testing in SOA context becomes easier since missing modules can be replaced by mocks. The paradigm "separation of concerns" becomes more naturally.

### 4.1.2. Interfacing

Of course, the most crucial point in realizing the Service Oriented Architecture is the module interfacing. The interoperability of modules is increased by the concept of message oriented middleware and loosely coupling (but like other architecture concepts it can never completely eliminate all difficulties rising from changes of interface definitions). The interface definitions are primarily not given by APIs but by protocol definitions in order to enable independent implementations of modules and communication APIs. The asynchronous communication paradigm (message) is preferred over synchronous communication (RPC) since it is more natural in a distributed environment and multi-service architecture where a reliable synchronous communication without need for timeouts is not possible.

### 4.1.3. Usage

The transport layer to send protocol items is realized by a dedicated message system. The protocol definitions for data exchange must avoid usage of BLOBs but instead must use self-describing data. The middleware and communication framework to fulfill those requirements is specified in [3].

The modularization of components must be guided as much as possible by the concept of Separation of Concerns, its realization, however, is domain specific.

Services should not persist client requests (e.g. subscriptions) in order to recover requested functionality in case of service shutdown or crash. Instead, a service API implementation used by clients should cache its client requests to retransmit in case of a service restart after shutdown or crash of the service.

### 4.1.4. API implementation

To ease the usage of data and message exchange for client applications, software services should implement APIs for specified programming languages on top of the message system and protocol definitions.

## 4.2. Data Persistence

If a software component demands means of data persistence it must be checked whether the provided functionality of the central Data Management System [4] already fits the needs. The Data Management System covers mechanisms, technologies, and guidelines how to use the central database management system of the FAIR control system. Usage of file system persistence should be avoided if possible.

## 4.3. Access Rights

Access rights (i.e. read or write access to settings and devices) are used to prevent conflicting changes of accelerator settings during operation, to prevent changes of global settings from local control rooms, or to prevent accelerator access for unauthorized personnel at all.

Currently (Aug. 2012), no exact guidelines to use access rights are specified.

#### 4.4. Error Handling

*Internal* errors are caused by a software malfunction. If such a malfunction is detected the software must generate log entries to help identifying error conditions. User applications must display human readable error messages and inform the user. Additionally, in case of unrecoverable errors, software on process level must generate trace information and must terminate. Software on library level (APIs) must not terminate the process.

*External* errors are caused by non-expected but possible external error conditions, e.g. by devices in offline-state or non-availability of service requests. Typically, such errors must not lead to program termination but must be logged. User applications must display the error and – if possible – must support the user to identify the error condition. Non-interactive services must inform clients in case of external errors during processing of client requests.

Generally, error messages should not be hard-coded in software but managed by and retrieved from a central message utility.

#### 4.5. Logging

Software logging, i.e. human readable text messages for software debugging and diagnostic, must utilize whenever possible the Diagnostic Logging Service functionality which is specified in [8]. An implementation of APIs must use the facility field of the logging system.

#### 4.6. Software Configuration

Usage of software configuration must use whenever possible the configuration service functionality which is specified in [10].

#### 4.7. Technology Constraints

The programming languages currently supported by the Accelerator Controls and Electronics Department are Fortran77/90, C/C++, Java, Python. But the software development for the FAIR service and application tier must use Java SE 1.8. Especially, the usage of Java Enterprise Beans is not accepted.

#### 4.8. IT Infrastructure Constraints

The current IT infrastructure constraints for the service and application tier of the FAIR accelerator control system are described in detail in [5][6][7]. Currently it can be expected that the Operating System is Linux (Scientific Linux 6.x). The computing environment is set up by a cluster of multiprocessor based computers connected by 1 Gb/s network. The GUI-environment is based on X-Windows architecture with remote displays. The source code of all software components must be accessible via version control system and ready to build from scratch by the Accelerator Controls and Electronics Department.

## I. Related Documentation

List of abbreviations for controls (Abbreviations\_Controls.pdf).

## II. Related Documentation

- [1] F-CS-C-01e, FAIR Common Specification "Accelerator Control System"
- [2] F-DS-C-01e, FAIR Detailed Specification "FEC Software Framework (FESA)"
- [3] F-DS-C-19e, FAIR Detailed Specification "Middleware and Communication Framework"
- [4] F-DS-C-24e, FAIR Detailed Specification "Data Management"
- [5] F-DS-C-25e, FAIR Detailed Specification "Controls IT Environment, part 1 (computing)"
- [6] F-DS-C-26e, FAIR Detailed Specification "Controls IT Environment, part 2 (network)"
- [7] F-DS-C-27e, FAIR Detailed Specification "Controls IT Environment, part 3 (network)"
- [8] F-DS-C-10e, FAIR Detailed Specification "Diagnostic Logging System"
- [9] F-DS-C-04e, FAIR Detailed Specification "User Applications"
- [10] F-DS-C-30e, FAIR Detailed Specification "Extended System Services"
- [11] F-DG-C-02e, FAIR Development Guideline "GUI Guideline"
- [12] F-DG-C-01e, FAIR Development Guideline "FESA Development Guideline"

## III. Document Information

### III.1. Document History

Version	Date	Description	Author	Review / Approval
0.1	03. Jan. 2012	Draft	R.Huhmann	R.Huhmann
1.0	31. Aug. 2012	Final Version	R.Huhmann	CCT