GANIL proposal for software architecture of the GET system

Authors : Frédéric Saillant, Bruno Raine, GANIL

1. Introduction

The aim of this document is to describe the general architecture of the software of the data acquisition system for the GET project and to define the different components of the system.
The objective of the task is to provide a data acquisition software, composed of tools usable from the debugging of boards by hardware engineers to the whole data acquisition system. It is important that the same tools be used during the development phase and to build the final system.
As the different boards of the system will be developed in different laboratories, the software delivered will have to be easily configurable to be adapted by the different teams.
As this electronic will have to be used in different laboratories, a particular attention will be taken in defining the frontier and interfaces between the GET system and local data acquisition systems of different laboratories (GANIL, MSU, GSI, …).

The GET software DAQ system has basically 4 main components :

- Slow Control which is in charge of setting up and monitoring the front end electronics (FEE) : AGET ASICs, ASAD boards, COBO boards, INBO, MUTANT and BEM.

- Run Control whose the main purpose is to control and monitor the software DAQ components. It coordinates the several activities necessary to put the GET system and its data acquisition software into operational state and to monitor it.

- Data Flow which concerns all the elements to process the data flow which is coming from the detectors up to the data storage or to DAQ systems of different accelerator laboratories.

- Experiment data base which contains all the description of the system

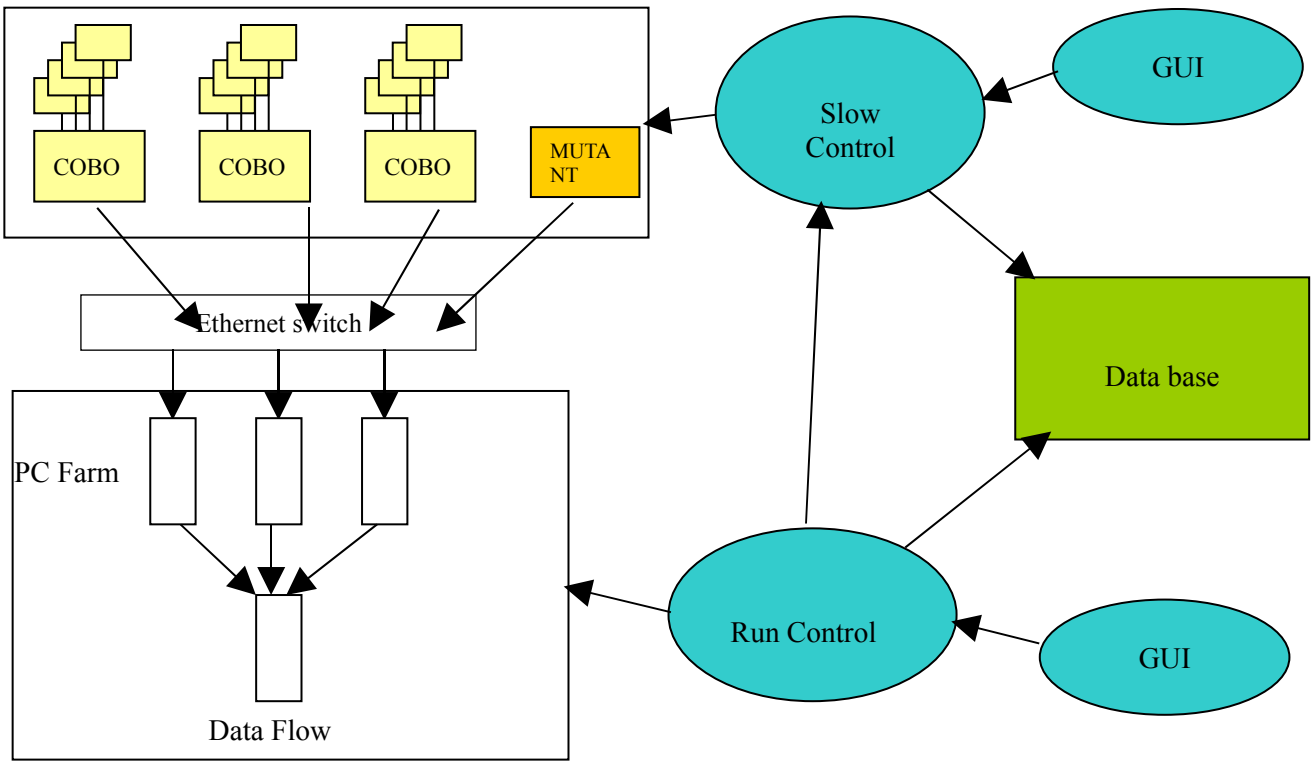

Figure 1 : General architecture

2. Slow Control

Slow Control is in charge of setting up and monitoring the electronics : AGET ASICs, ASAD boards, COBO boards, INBO, MUTANT and BEM.
The Slow Control system will be designed with a client/server approach. It will mainly consist in a Slow Control Core (SCC) communicating on one side with the different boards and on the other side with a graphical user interface which will behave as a client of the Slow Control Core.

2.1.Slow Control Core

The main functionalities Slow Control has to provide are :

- define which electronic boards are present in the system
- initialize the different boards with correct values
- save all the setup parameters of the whole electronics or a part of the electronics
- restore a previously saved setup
- monitor some key parameters of the boards (temperatures …)
- handle error/alarm events and pass them to the Run Control
- accept some basic commands from the Run Control (setup, go, stop, get state …)

From the GANIL point of view, the communications between the Slow Control Core and its different interlocutors (external graphical interface, embedded software, run control,…) could be implemented with the SOAP/XML network protocol and a WSDL (Web Service Description Language) interface.

The architecture of the Slow Control Core could be designed in such a way that it will be reusable in other projects.

2.2.User interface

A graphical user interface will be provided to access the functionalities of the slow control core.

Several levels of interface have to be provided:

- ✓ A basic level which gives access to registers of the boards. This is very useful during the development phase of electronic boards and for debugging during exploitation of the system.
- ✓ An expert level, for access to some parameters which have to be accessed only by specialists.
- ✓ A user level giving access only to needed parameters to configure an experiment.

2.3.Embedded software

Having in mind to put as much intelligence as possible inside the electronics, boards containing a Xilinx Virtex FPGA with a PowerPC core will embed a network command server which provides advanced network services (TCP/IP, HTTP, SOAP…).

This command server will have to provide very basic accesses to the individual registers of the hardware to enable easy debugging of the boards.
More sophisticated functionalities will be implemented to allow an object approach of the application.

GANIL team proposes to use the ENX command server designed for the AGATA project (see http://enx.in2p3.fr). This server provides basic access to registers with a well defined interface using

SOAP protocol. The interface with hardware is made by an intermediate driver level, which can be dynamically loaded by using shared libraries. The "special function" enables to enlarge functionalities as needed for high level access. This software has been written in ADA. The drivers can be developed in ADA or C++.

3. Run Control

The Run Control has the main purpose to control and monitor the software DAQ components. It coordinates the several activities necessary to put the GET system and its data acquisition software into operational state. Actions like initialization, setup, start and stop of the data acquisition are performed by the operator through the Run Control system. It interacts with the Slow Control system in order to assure the correct configuration and setup of the electronic devices, before data taking is started. It also provides the monitoring of the acquisition, error report and logging capabilities.

The main tasks are outlined here :

- configure the DAQ software system for a run by selecting data flow active components
- save/restore a configuration
- basic commands to control all the active components (setup, start, stop …)
- monitor the DAQ (status, data rates …)
- handle error/info messages
- log book

As the Slow Control, the Run Control system will be designed with a client/server approach. It will also consist in a Run Control Core (RCC) communicating on one side with the data flow active components and the Slow Control Core, on the other side with a client graphical user interface. Whenever it will be possible the communications will be implemented with the SOAP/XML protocol and WSDL interface.
It is proposed that the RCC will be designed on the schematic of the current one being developed at GANIL in a general purpose way.

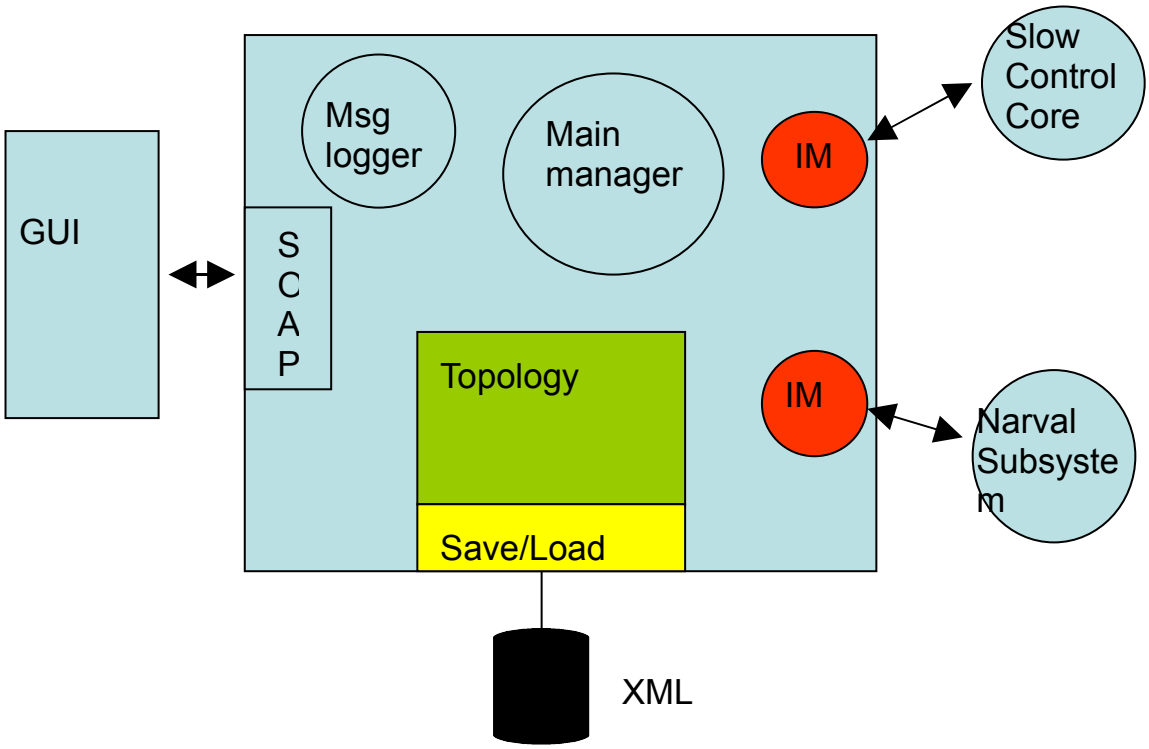

Figure 2 : Run Control general view

This Run Control is designed to associate and control parts of DAQ which have been designed by different partners and do not support the same set of commands, communication protocols, etc… For this purpose, it is designed around the concept of "Instrument Managers".

The Instrument Managers allow to remote control and monitor the DAQ elements – called the "instruments". Each Instrument Manager controls a unique instrument. The Instrument Manager performs the actual communication with an instrument, thus acting as a protocol adapter that implements the instrument specific protocol to access its functions and read its status. A top Manager coordinates the whole system.

As the DAQ elements are linked together by data flow links, the Instrument Managers must be organized in the same way. The identification of the instruments and the data flow relationships between them constitutes the topology of the DAQ system. This topology can be described thanks to a set of Web services and an available graphical user interface which uses these Web services. The topology can also be saved on disk in order to be reloaded at any time. For this purpose a description language based on the XML standard has been developed by GANIL.

The Instrument Managers identified in the GET project are :

- ✓ Slow Control Core
- ✓ A Narval subsystem

The Message Logger also receives all the information/error messages generated by the instruments or by the Run Control Core itself. It allows to log them on a disk file with an XML format. The Log4cxx package is used to handle these messages. All the messages generated by RCC are logged with Log4cxx.

4. Data flow

The goal is to process the data flow which is coming from the detectors up to the data storage. Data sources are the detectors after their front-end electronics (ASAD boards). The ASAD outputs are collected by the COBO boards. Then all data coming out from the COBO boards are merged together thanks to an Event Builder taking into account the physics correlations provided by the MUTANT board.
Considering the large data rate, the event builder can provide a level 3 trigger to reduce data. In this case, it can be implemented in a computer farm, depending on the power needed for filtering. The distribution of parts of event from Cobo boards will be made by using time slice principle, so that a computer will receive all parts of a same event.
At the end of the chain, data have to be provided to the physicists in an understandable and user-friendly way thanks to a Data Format library, and to be stored on a large disk array or to be sent to specific DAQ of different accelerator rooms.

The data flow can be based on the Narval data acquisition system currently used at GANIL. It is a distributed and entirely configurable software which can be easily adapted to the different configurations, especially for the event builder (data merger, data filtering) of the GET system.

Narval is composed of a set of processes (actors) handling the acquisition's data flow. In order to control and coordinate the system, all information about topology, configuration, state machine, are stored in a process named coordinator.
An actor is mainly a process that takes part of handling the acquisition's data flow. There are four kind of actors :

- Producer : injects data in a Narval system
- Intermediary : acts like a NxM switch or data filter
- Consumer : end of line actor, typically used for storage
- Standalone : exception to the rule, this actor don't handle data but is aware of the system state machine.

Data transfert between actors is handled through TCP/IP or Infiniband.
Narval has been written in ADA. It is easily linkable with C++ to implement specific algorithms in actors.

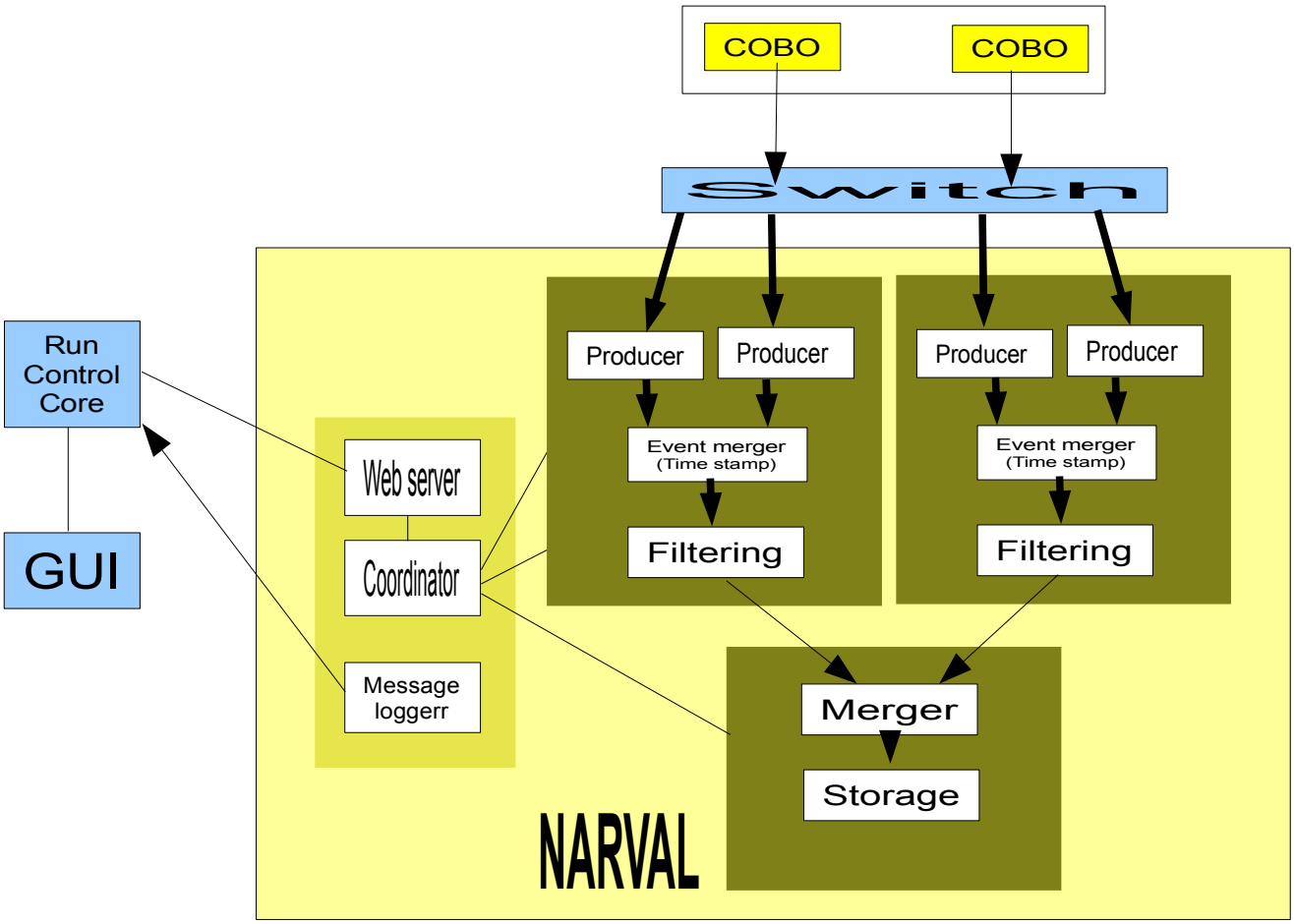

Figure 3 : Example of Narval topology for GET

The different items to be developed are :

- ✓ Embedded readout process in Cobo
- ✓ Narval actors
  - ✓ Data mergers
  - ✓ Data filtering (level 3 trigger)
- ✓ Data format

5. Data base

The description of the configuration has to be saved in files or in data base…..