SPIRAL2 DAQ ICC working group first recommendations

S. Anvar, X. Grave, E. Legay, E. Liénard, A. Ordine,

S. Panebianco, B. Raine, F. Saillant, H. Simon

1 Introduction

The objectives of the ICC DAQ working group are to make recommendations and to define standards for the development of the DAQ software for existing and future detectors of GANIL-SPIRAL2. This group is composed of one representative per equipment.

The general architecture of the data acquisition system for GANIL-SPIRAL2 has been discussed and approved.

The aim of this document is to describe the architecture of the GANIL-SPIRAL2 data acquisition system, to define interfaces between the different parts of the system, and to give rules and recommendations for external system to connect to the GANIL-SPIRAL2 system.

Considering the requirements of the SPIRAL2 detectors, a chapter is dedicated to the needed GANIL-SPIRAL2 infrastructure in terms of network and storage.

2 GANIL-SPIRAL2 DAQ requirements

Regarding the existing and future detectors for SPIRAL2, the GANIL-SPIRAL2 data acquisition system will have to be highly modular to accept small scale to large scale experiments.

Considering the complexity of the future detectors for SPIRAL2, most of them will have a dedicated electronics. Some of them will be located at GANIL, but most of them will be used in different laboratories. So it is important to consider the concept of DAQ subsystems which can be interconnected when detectors are associated together.

Most of the new detectors will provide high data flows from different branches which will need online event building to merge the branches and filtering to reduce the amount of data to be stored.

For some of the detectors, an evaluation of data rates and of amount of data to be stored is given in table1.
It is shown that the bandwidth required for SPIRAL2 experiments can be very high. These values have to be confirmed by the collaborations. For instance, up to now, AGATA has stored raw data with a configuration of 5 triple clusters on their own disk. What will be the situation with 15 triple clusters?

	Data bandwidth in MBytes/s		Data generated in TBytes/day		Comment
	Raw	Filtered	Raw	Filtered	
AGATA	560	9	50	0,8	Evaluation for GANIL phase with 15 Triple Clusters
EXOGAM2	30 to 90		3 to 8		Depends on scenarios
NEDA	60 to 500		5,5 to 44		Depends on embedded compression algorithms and scenarios
ACTAR	200	20	17	1,7	
S3	40		3,5		

Table 1 : Previsionnal data bandwidth for SPIRAL2 detectors

3 Global architecture

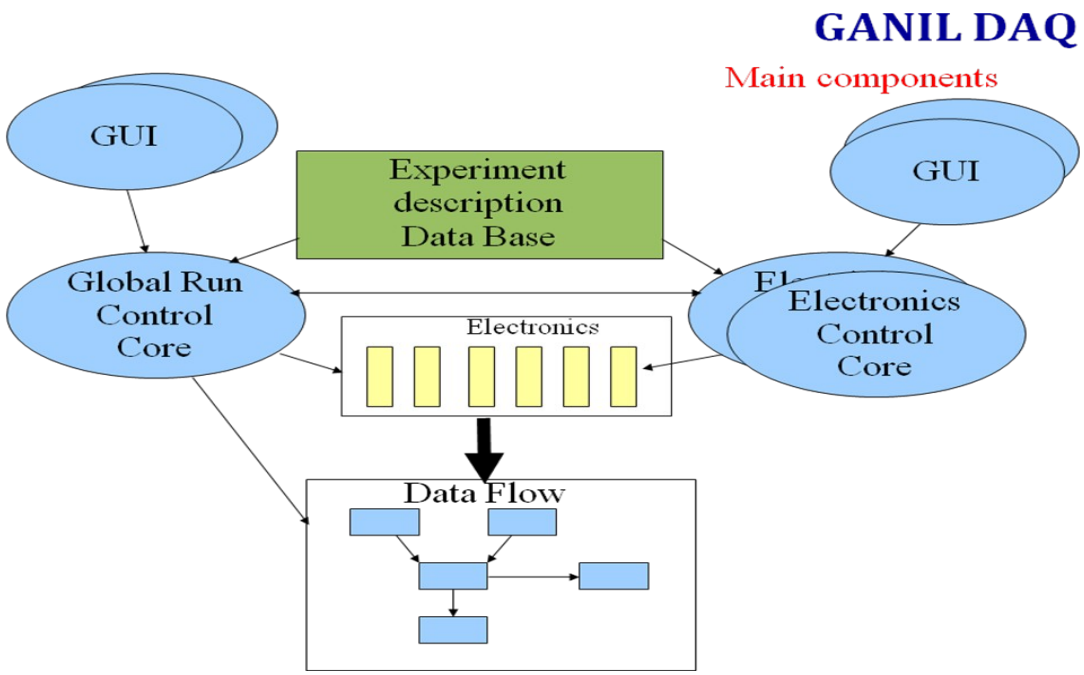


Figure 1: GANIL-SPIRAL2 DAQ architecture overview

The software DAQ system has basically three main functionalities :

- Electronics Control which is in charge of setting up and monitoring the electronic boards
- Run Control whose the main purpose is to control and monitor the DAQ components. It coordinates the several activities necessary to put the system and its data acquisition software

into operational state.

- Data Flow which processes the data flow which is coming from the detectors up to the data storage. At GANIL the Narval system has been chosen for the standard data transport.

All communications between parts of the system are made by using SOAP protocol.

3.1 Electronics Control

The Electronics Control is an application in charge of the complete control of the hardware. It has to know the description of electronics, to setup and monitor it during the experiment.

The Electronics Control system will be designed with a client/server approach. It will mainly consist in an Electronics Control Core (ECC) communicating on one side with the different boards and on the other side with a graphical user interface, the Run Control, calibration tasks, etc... which will behave as clients of the Electronics Control Core.

Electronics Control consists in two sub-tasks : slow control and electronics run control.

The main tasks of Slow Control are to :

- Describe hardware configuration
- Save / restore hardware configurations
- Setup boards registers
- Monitor boards (temperatures, voltages ?)
- Provide tools for hardware development and maintenance

The main tasks of Electronics Run Control are to :

- Manage the embedded readout process state machine
- Setup the embedded readout processes
- Dispatch commands from Global Run Control to the embedded readout processes
- Monitor the embedded readout processes
- Handle errors and pass them to Global Run Control

In a multi-system configuration several Electronics Control systems may be present.

3.2 Run Control

The Run Control has the main purpose to synchronize and monitor the software DAQ components. It coordinates the several activities necessary to put the 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 Electronics 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 (setup, start, stop …) to control all the data flow active components
- basic commands (setup, start, stop …) to control electronics through the Slow Control Core
- monitor the DAQ (status, data rates …)
- handle error/info messages
- log book

The Run Control system, currently under development at GANIL in a general purpose way, is designed with a client/server approach. It consists in a Run Control Core (RCC) communicating on one side with the data flow active components and the Electronics Control Core(s), on the other side with a client graphical user interface. The communications are implemented with the SOAP/XML protocol and WSDL interface.

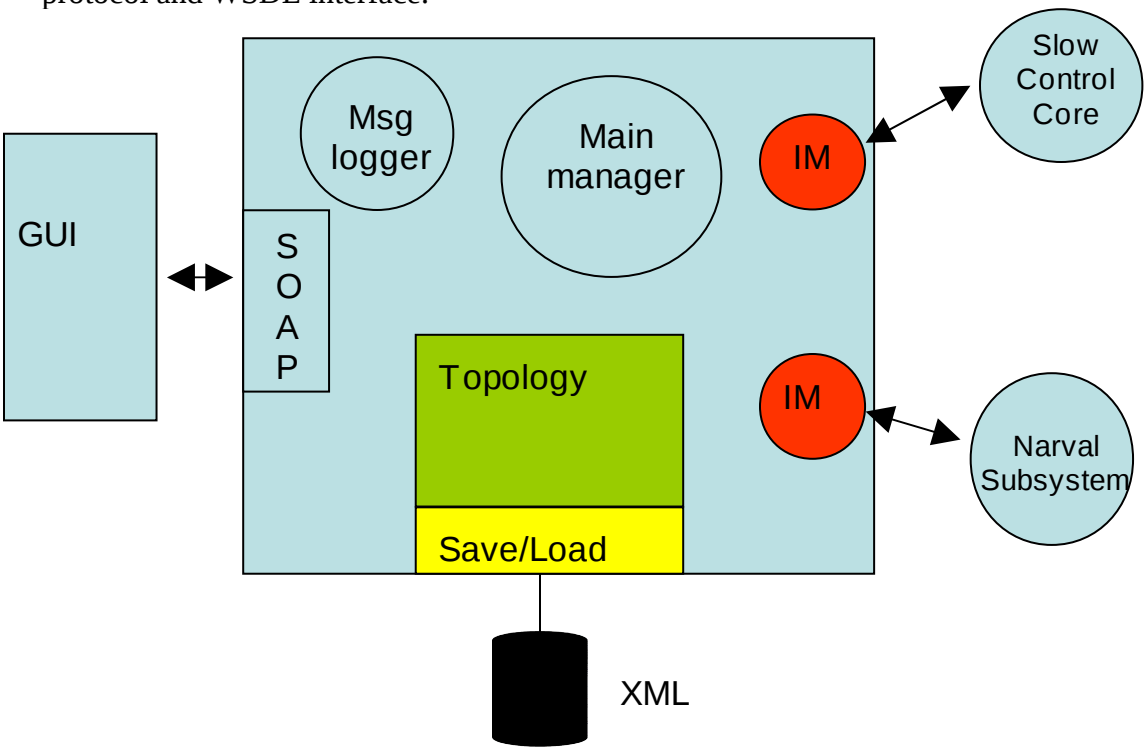


Figure 2: Run Control general view

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 constitute 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 GRCC provides two Message Logger services to receive all the information/alarm messages generated by the instruments or by the Run Control Core itself. One of the two services will be dedicated to alarm messages which request an action, the other one will only display and log messages. The Log4cxx package is used to handle these messages. It allows to log them on a disk file with an XML format.
The client applications have to use Log4xx mechanism (Log4j, Log4cxx, Log4ada,... depending on the language of the application) to propagate their messages to the central message loggers.

It is important to understand that the Run Control doesn't know the details of the electronics. This is the role of the Electronics Control Core (ECC) application. It only sends global commands to ECC like Init, Start, Stop,...

3.3 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. Considering the large data rate produced by SPIRAL2 multi-detectors, the Data Flow must provide all tools necessary to build a distributed data acquisition with a multi level of event building and filtering.

The data flow will be based on the Narval data acquisition system originally developed at IPNO and currently used at GANIL. It is a distributed and entirely configurable software which can be easily adapted to the different configurations.

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 kinds of actors :

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

Data transfer between actors is handled by using UNIX FIFO in a same computer and through TCP/IP or Infiniband between computers. A new version under development will use shared memory for data transfer in a same computer.
Narval has been written in Ada. It is easily linkable with C++ to implement specific algorithms in actors.

The architecture of this framework enables easy interconnection with other DAQ using or not using Narval.

4 Rules and Recommendations

4.1 Who does what?

- The **Global Run Control** is provided by GANIL
- **Narval general data flow** is provided by GANIL
- **Specific Narval actors** are provided by collaborations. They can be developed in the form of C++ or Ada libraries.
- **Specific Electronics Control** systems are provided by collaborations taking into account the interface specifications defined by the GANIL and the SPIRAL2 ICC DAQ working group.

4.2 Rules

To integrate an equipment in the GANIL-SPIRAL2 DAQ system, a minimum set of rules has to be followed.

4.2.1 Electronics Control

The electronics control application must implement a minimum set of Web services, a minimum state machine and a backup of the configuration. Moreover, it may implement error reporting to a centralized message logger.

See the document **"ECC state machine & WSDL services"** in Annex 1 for the details of implementation of the ECC interfaces.

4.2.1.1 State machine

A minimum state machine has to be implemented. The following figure shows the current version, still under discussion. See more details in Annex 1

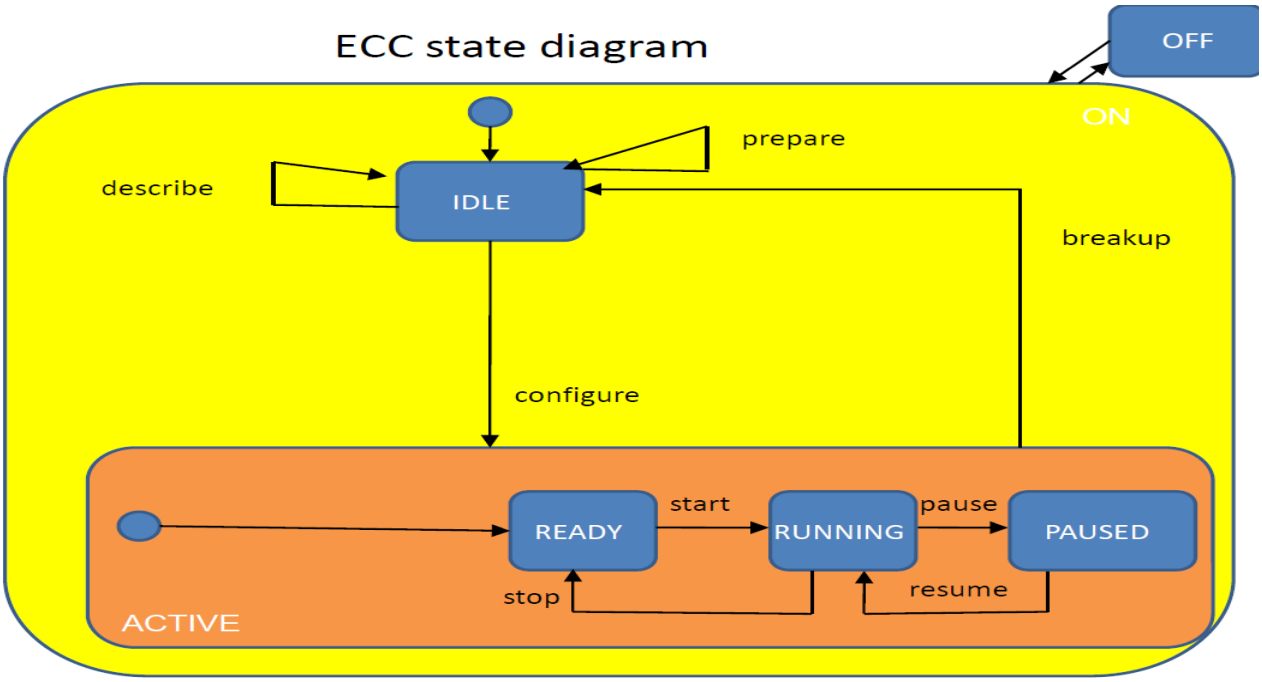


Figure 3 : ECC State Machine

4.2.1.2 Web services

A minimum set of Web services based on XML/SOAP to remotely access an Electronics Control Core (ECC), typically from GANIL Global Run Control Core (GRCC) must be implemented. The procedures made available via the SOAP protocol allow for control of the ECC state machine and for monitoring of the ECC current status commands using SOAP protocol.
The list of proposed Web services is given in Annex1

4.2.1.3 Backup

A run number is a concept associated with a configuration of the all parts of the DAQ at the beginning of the run. It is recommended to associate the current backup of the configuration to each run. This can be done by serialization of the data bases in files associated with the run.

As far as possible, provide a C++ library to access parameters of the configuration. Abstract functions have to be defined.

4.2.1.4 Message logging

ECC has to propagate its messages and alarms by using the Log4xx mechanism(Log4j Log4cxx, Log4ada,... depending on the language of the application) , as previously described in §3.2

4.2.2 Data format

It has been agreed to have a standard data format for GANIL-SPIRAL2 runs based on the **Multiframe MetaFormat** (MFM2.2) described in Annex 2.
The main goal of the MultiFrame metaformat is to be able to define binary formats for data acquisition and serialization that are self-contained, layered, adapted to network transfers and evolving.
For new equipments, it is recommended to base the description of data on the MFM.
For existing equipments, whose data format is already frozen, it is possible to include a filter in the data flow to encapsulate data in an MFM data frame or to adapt the data format to MFM.

5 GANIL-SPIRAL2 infrastructure

Regarding the requirements of the different collaborations, it appears that the data flow may vary from 10 to more than 500 MBytes/sec, depending on the equipment. A data rate of 100MBytes/sec means a storage of 8TBytes/day, or ~ 100TBytes/experiment.
Even if some optimizations can be done on trigger conditions and data compression, Ganil will have to provide the network and storage infrastructure to accept data flows of the order of several hundreds of MBytes/sec. This first storage has to be seen as cache storage.

Considering the large amount of data produced by the Spiral2 detectors, the collaborations must provide a solution to store the data in a long-lasting way. They could have an account in a computing center like CC-IN2P3 to backup their data. Ganil has to study solutions to provide

enough local cache disk and bandwidth to backup data to CC-IN2P3.

PC farms are needed by most of the collaborations for event building and data reduction. Two possibilities have been mentioned :

- The collaborations bring their own configurations. In this case Ganil will have to provide rooms with cooling
- Ganil provides PC farms which can be attributed to experiments

In summary, GANIL has to initiate a study for its infrastructure on the following points

- local network : 10Gbits network distributed in experimental area
- local storage : several hundreds of TBytes extensible
- Long-lasting storage : possibility to send data to a computing center like CC-IN2P3, by using a black fiber between Ganil and CC-IN2P3
- computers : workstations for experiment control and data acquisition, including as far as possible a set of PC farms

6 Conclusions

A general architecture for the data acquisition system of GANIL-SPIRAL2 has been approved by all participants of the working group. This architecture is in accordance with all the current detectors under development.

Considering the different laboratories involved in the detector projects, it appears that it will be difficult to use the same tools in all the projects for the setup of electronics and the backup of configuration.

A set of rules and recommendations have been enacted to enable easy integration of a specific data acquisition system in the GANIL-SPIRAL2 DAQ.

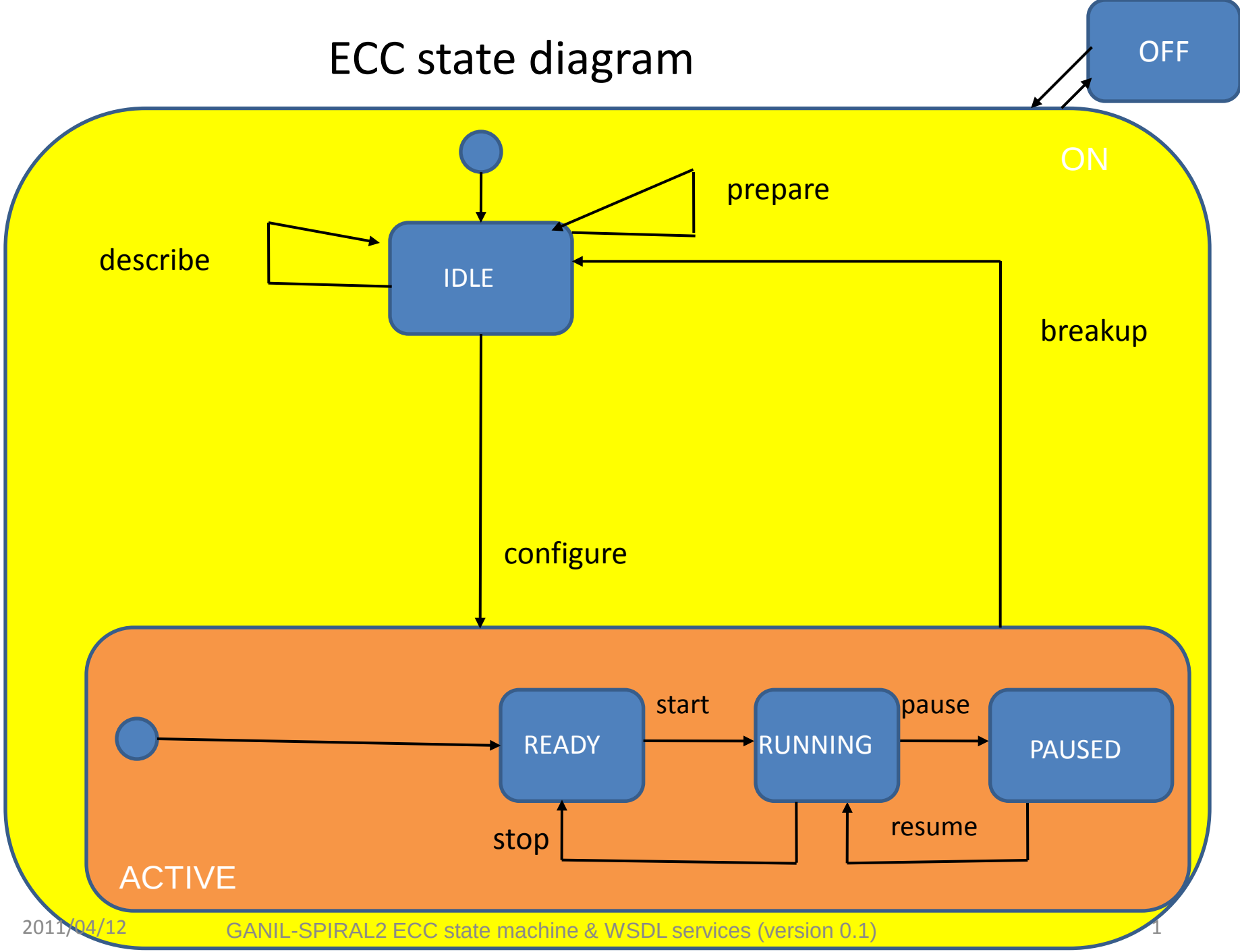
GANIL has to study solutions to provide adapted infrastructure in term of network, storage and computing.

The current GANIL data acquisition system has been using the previously described architecture with Narval and GRCC since the beginning of 2010. A lot of work has to be done to provide all the functionalities described, but it has been shown that this architecture is really modular and enables easy coupling of different detectors. Current experiments use a bandwidth of a few MBytes/sec. Tests of performance have to be done at GANIL to reach several hundreds of MBytes/sec.

Annex 1

ECC State Machine & WSDL services

ECC state diagram



ECC states description

State	Description
IDLE	no configuration is loaded, in particular no physics parameter has been loaded in the boards
READY	• configuration has been applied on hardware • acquisition threads have been activated and connected to their corresponding Narval actors
RUNNING	the run is in progress, hardware is delivering data ...
PAUSED	the run is paused, hardware is not delivering data, but the run conditions will not change when we resume

ECC transitions description

Transition	Description
describe	setting up hardware topology (which boards are in the system ...)
prepare	inter-board communications, clock synchronizations, calibrations ...
configure	loads physics parameters into the registers of the boards (thresholds ...)
start	starts the run, i.e. allows hardware to deliver data
pause	• pauses the run, i.e. forbids hardware to deliver data • leads to PAUSED state
resume	resumes the run, i.e. allows again hardware to deliver data
stop	• ends the run, preventing hardware from delivering data • leads to READY state
breakup	• invalidates physics parameters setup in the boards • leads to IDLE state (so physics parameters will have to be setup again)

Numeric values for ECC states

State name	Value
OFF	0
IDLE	1
READY	2
RUNNING	3
PAUSED	4

Numeric values for ECC transitions

Transition name	Value
NONE	0
DESCRIBE	1
PREPARE	2
CONFIGURE	3
START	4
STOP	5
PAUSE	6
RESUME	7
BREAKUP	8

Numeric values for errors

Error	Value
no error	0
generic error	1
error during describe	2
error during prepare	3
error during configure	4
error during start	5
error during stop	6
error during pause	7
error during resume	8
error during breakup	9

WSDL interface proposal for ECC

Functions :

- GetConfigIDs
- Describe (configID)
- Prepare (configID)
- Configure (configID, table)
- Start
- Stop
- Pause
- Resume
- Breakup
- Special
- GetStatus

Annex 2

The MultiFrame Metaformat Specification

The MultiFrame Metaformat Specification (2.2)

S. Anvar, R. Fox, B. Raine, F. Saillant, N. Usher

1 Goal and Features

The main goal of the MultiFrame metaformat (MFM) is to be able to define binary formats for data acquisition and serialization that are self-contained, layered, adapted to network transfers and evolving.

- Self-containment is achieved through the use of in-frame metadata information.
- Layering is achieved by allowing a data frame to include other data frames.
- Network adaptation is achieved by allowing limited necessary decoding (basically byte counts) for transmission purposes.
- Extensibility is achieved by including byte counts for frame elements (e.g. Headers), version fields and additive extensibility rules that allow both backward and forward compatibility with user software.

2 Format Specification

2.1 Numbers and Sizes

All sizes are in **BYTES** unless specified otherwise, a BYTE being an 8-bit information unit.

By default, all variables and fields are organized as Big Endian variables (most significant byte comes first). However, an endianness bit is foreseen as part of the Frame metadata.

2.2 Definitions

A "Frame" is a self-sufficient, continuous set of bytes composed of two sections:

1) A "Header" section,
2) A "Data" section.

2.2.1 The Data Section of a Frame

The Data section of a frame is composed of "Items" *of the same type* and a possible reserved ending section called the "Data Reserve." There are 3 basic types of Items:

1) The "Fixed size" Item,
2) The "Variable size" Item,
3) The "Blob" Item

A Fixed size Item is a block of data made of a fixed number of bytes. Typically, a C structure would appear as a Fixed size Item. A Frame whose Data section contains Fixed size Items is called a "Basic Frame." Such a Basic Frame contains any number of Items provided that they are all of the same size and appear consecutively in the byte stream of the Frame.

A Variable size Item corresponds to a block of data whose size is variable. In our case, *it is implemented as a Frame*, i.e. any Variable size Item is itself a Frame as defined by this specification.

A Blob Item corresponds to an opaque block of data whose internal structure is not described in the Header.

2.2.2 The Header Section of a Frame

The Header section of any Frame contains the information describing:

1) The endianness, "blob-ness" and unit block size (a power of 2 number of bytes) of the Frame .
2) The total size of the Frame *as a number of unit blocks*.
3) The data source id referring to the (type of) apparatus that has produced the Frame data.
4) The Frame type, encoding both the Header type and the data type that constitute the Frame.
5) The revision number for that Frame type.

The Header section of a non-blob Frame contains the former fields followed by:

6) The total size of the Header itself *as a number of unit blocks*.
7) The byte size of the Frame Item, size 0 corresponding to Variable size Items.
8) The number of Items contained within the Frame.

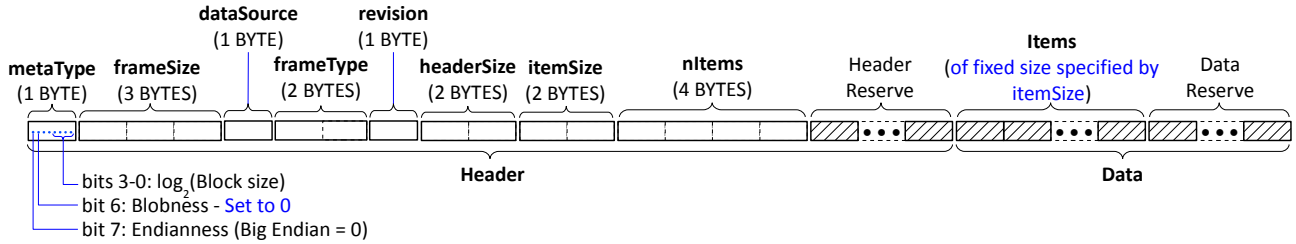
The Header Section of a non-blob Frame can also have an ending section called the “Header Reserve”.

2.3 Detailed Structure

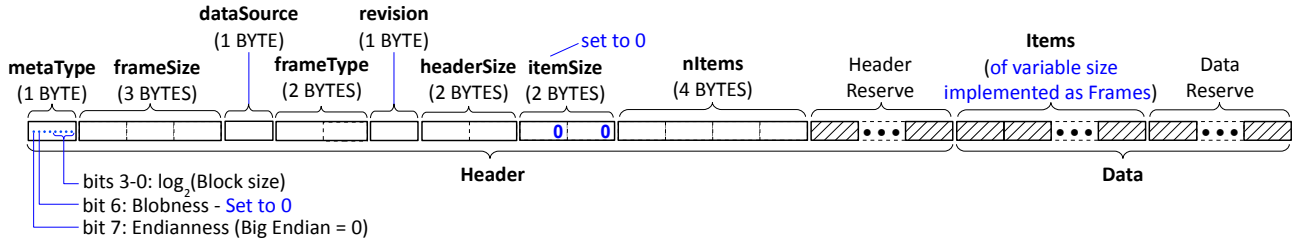
Blob Frames are characterized by a minimal Header and an opaque Data part. As a consequence, they do not address the software compatibility issues associated with the evolution of Header formats within a given application. For this reason, Basic and Layered Frames are recommended over Blob Frames. The use of Blob Frames should be reduced to frame types whose structure are not well adapted to Basic or Layered Frames, for instance, when the Data part of a frame type is too small as compared to the Header, so that a non-blob Header would be too much of an overhead.

2.3.1 The Structure of the 3 Frame Meta-types

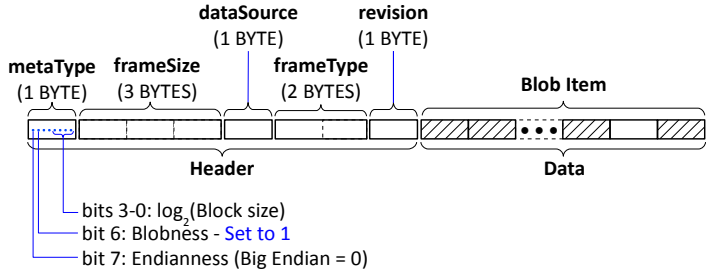
Basic Frame



Layered Frame



Blob Frame



2.3.2 Semantics of the Frame Fields

Field Name	Field Size	Field Description
metaType	**1 BYTE**	**`ISLEND`** ≡ Bit 7: endianness. Set to 1 if Frame fields are Little Endian. **`ISBLOB`** ≡ Bit 6: "blobness." Set to 1 if Frame is a Blob Frame. **`P2BLCK`** ≡ Bits 3-0: Unit block size (in bytes) as a power of 2. The frame size and header size are aligned on this.
frameSize	**3 BYTES**	This field represents the Frame size *expressed as an integer number of unit blocks*. This size is the addition of the sizes of the Header section and the Data section.
dataSource	**1 BYTE**	This field represents the source from which the data of this Frame were produced. It would typically be used to refer to the detector or detector type from which the Frame data have been extracted.
frameType	**2 BYTES**	This field is meant to receive the type of this Frame: Frames differ in type if their Items or Header or both differ in any way that breaks revision rules.
revision	**1 BYTE**	This field is meant to receive the Frame format revision: Two formats which differ only by revision have the same type and are both forward and backward compatible (e.g. old software based on format 2.3 can partially read format 2.5 without failing).
headerSize	**2 BYTES**	This field contains the total size of the Header section *expressed as an integer number of unit blocks*. The Header section consisting in the fields: *metaType, frameSize, dataSource, frameType, revision, headerSize* itself, *itemSize*, *nItems* and a possible "reserve" sub-section for the Header.
itemSize	**2 BYTES**	• set to 0 for layered Frames, i.e. when the Items are themselves Frames; • set to the Item size *in bytes* when the Frame is constituted of fixed size Items.
nItems	**4 BYTES**	This field receives the total number of Items in the Frame. WARNING: *itemSize* × *nItems* is not necessarily equal to the full size of the Data section; it can be smaller if the Data section ends with a "Data Reserve" sub-section.

2.3.3 Additional Remarks

The "reserve" sub-sections at the end of both Data and Header sections of non-blob Frames are not mandatory. They have been provided for two main purposes:

1) The user can extend the Header section in order to include application-specific fields, provided that the value of the *headerSize* field takes the extension into account. For any software that is not aware of these extensions, they appear as a mere "reserve" section to be skipped. See next section for format revision rules that allow backward and forward compatibility between different revisions of decoding software in an application.
2) The user can add padding in the Reserve subsection for alignment purposes or other low-level constraints typically encountered on embedded/real-time platforms. In particular, if the total byte size of the Data section or the Header section and its possible application-specific extensions do not add up to an exact multiple of the unit block size defined (as a power of 2) in the *metaType* field, the section must be padded accordingly (remember that the *headerSize* and *frameSize* fields are expressed as a number of unit blocks).

As explained in the Frame fields semantics table, each frameType refers to a unique couple of Header type and Item type. Consequently, two different frameTypes may refer to the same Item type and differ only because of the Header or vice-versa. However, the Header section of a Blob Frame is not allowed to evolve.

The size of an Item type as implemented by an application may be strictly smaller than the size indicated in the itemSize field, allowing any fixed Item to include trailing "reserved" bytes. This situation particularly arises when the actual Items of the frame belong to a revision that is higher than the revision for which the application was compiled (see Format Revisions further).

3 Format Revisions

The MFM has been designed with the purpose of allowing easy evolution in data formats so that users can provide for software and system enhancements without the need to force all related software to evolve at the same time. However, the feature can be implemented only if, from the beginning of software/hardware development, the following Format Revision Rules are enforced in Data format definitions for all Frame meta-types and and in Header format definition for all non-blob Frames.

3.1 Format Revision Rules

1) Format revisions can be implemented for both Header and Data sections in Basic and Layered Frames, but only in Data section for Blob Frames.
2) Format revisions apply only to application specific formatting: no change is allowed for the fields defined by MFM in section 2. In particular, the field structure *metaType, frameSize, dataSource, frameType, dataSource, revision, headerSize*, *itemSize* and *nItems* must remain untouched.
3) An increase in format ***version*** means that at least a new ***frame type*** has been added to the format, implying that old software will not be able to decode some Frame types in the new format, although it will be able to skip, store, transfer or even split such Frames, basically using all size information in the Header. Such changes must increment the format version, e.g. 2.6 to 3.0.
4) An increase in format ***revision*** means that new fields have been added (to Header or Item), former fields remaining identical to former revision, e.g. old software implementing format 2.6 will be able to decode (partially but coherently) any data in format 2.8.

4b) More precisely, in the case of a revision change, the format extension cannot remove any already defined field, nor can it modify its offset relative to the beginning of its section (Header or Data); in other words, a new field cannot be set to occupy bytes that are already occupied by a former field.

3.2 Additional Remarks

If the format revision rules are respected, then an old software that is able to decode version ***n.x*** data will still be able to safely decode version ***n.(x+1)*** data: to this old software, the new fields will be easily skipped as they will appear as “garbage” within the Reserve sections.