



# Merge of ancillary detectors with AGATA. Interaction with the Run Control and Data Analysis.

## Specifications.

N. Dosme<sup>2</sup>, X. Grave<sup>1</sup>, E. Legay<sup>2</sup>, V. Pucknell<sup>3</sup>, F. Saillant<sup>4</sup>, Ch. Theisen<sup>5</sup>

<sup>1</sup>IPN Orsay

<sup>2</sup>CSNSM Orsay

<sup>3</sup>Daresbury

<sup>4</sup>GANIL

<sup>5</sup>CEA Saclay

Draft version 0.5, January 2006

## 1. Introduction

This document describes the interface between AGATA ancillary detectors and AGATA acquisition systems.

Estimates of the typical data rates expected after the coupling of ancillary detectors with AGATA and the AGATA demonstrator are described. Ancillary detectors already foreseen for AGATA are taken as an example, which shows how an ancillary detector reduces the data flow after the pre-processing level.

The data merging procedures using the event-number or the clock counter correlation modes are considered. A scheme for the interaction of the ancillary data acquisition and NARVAL is proposed: identification and data exchange protocol. Since the merge process depends on the data structure and contents, the implementation of libraries is proposed.

In the last part of this document, the interactions with the Run Control, tracking and Data Analysis are briefly discussed.

## 2. Rates

To avoid any confusion, the following terminology is used: a hit correspond to a signal and the associated data in a single detector (for instance an AGATA segment, a Si detector element); an event corresponds to a set of hits corresponding to a trigger condition.

Numbers given in this section are estimates and should be updated by the ancillary detector users.

## 2.1. AGATA rates

The maximum hit rates taken for the AGATA demonstrator and AGATA without any additional conditions from ancillaries or  $\gamma$  multiplicity are:

AGATA demonstrator:

Single rate = 14 kHz	Total rate = 83 kHz
----------------------	---------------------

Full AGATA:

$M\gamma = 1$	Single rate = 15 kHz	Total rate = 3 MHz
$M\gamma = 30$	Single rate = 50 kHz	Total rate = 300 kHz

Note that  $M\gamma = 1$  corresponds mainly to coulomb excitation experiments. An ancillary device to detect scattered beam and/or target is mandatory, but has not been yet considered by the AADWG. This ancillary detector should reduce the triggered rate and the data flow after the pre-processing level by a significant factor.

## 2.2. Ancillary detector rates

Worst data rates are considered in this section.

### 2.2.1. Gamma array

The coupling of gamma arrays like Clara, Exogam, Jurogam or Rising is foreseen for the demonstrator phase. A maximum un-triggered hit rate of  $\sim 100$  kHz is expected. Setting trigger conditions including ancillary detectors as shown below will scale down the rates. The reduction will be similar to that of the demonstrator.

Jurogam is a particular case since it is part of the TDR trigger-less system. This coupling will be considered separately in paragraph 3.2.

#### 2.2.1. Spectrometers: FRS, Prisma, RITU, Vamos

The event rate at the focal plane of a spectrometer is in the order of a few kHz. Focal plane detectors will be included to the AGATA (demonstrator) trigger and will be a necessary condition to accept AGATA data. Each event may include a few hundred hits, most of them being empty.

The triggered event rates for both spectrometer and AGATA will be therefore a few kHz.

#### 2.2.2. The neutron wall

The neutron-wall un-triggered event rate is expected to be similar to the gamma un-triggered event rate. Each neutron wall event includes a few hits. Trigger condition is usually a combination of the neutron-wall and gamma multiplicity, resulting in a reduction of a factor  $> 10$  compared to un-triggered rate.

#### 2.2.3. CUP

CUP un-triggered rate in the same order of the gamma event rate. CUP is usually used in coincidence with the neutron-wall, resulting in a reduction of a factor  $\sim 10$  of triggered events. A Cup event includes a single hit.

### 2.2.4. Charged particle prompt detector

Similar to the neutron wall: un-triggered rate similar to un-triggered gamma rate. A reduction of a factor  $> 10$  is expected after setting trigger conditions. A Charged particle detector includes a few hits.

### 2.2.5. RFD

The raw RFD rate is extremely high: 1-10 MHz per detector module. The RFD triggered rate is reduced using coincidence with the accelerator HF signal and with the gamma multiplicity. Triggered rates for both RFD and AGATA are in the order of a few kHz. In principles, an RFD event includes a single hit.

**A minimum reduction of 10 in the triggered data rate is expected using an ancillary detector.**

**In the worst case, the following rates are expected:**

**Triggered AGATA rate  $\sim$  triggered ancillary rate  $< 1/10$  un-triggered AGATA rate,  
with un-trigger rates shown in paragraph 2.1.**

Computing power for merging of ancillaries corresponds therefore to  $\sim 1/10$  of the AGATA standalone merge computing power. Using clock counter correlation with the TDR system should not increase significantly the power requirements, since the counting rate is expected to be rather low.

## 3. Correlation modes

### 3.1. Event-number correlation

It is not expected to check whether the clock counter is consistent with the trigger conditions. Only event-number should be used to merge data.

### 3.2. Clock counter correlation with the TDR system

The TDR will produce 4 types of data, corresponding to 4 different processing. TDR doesn't produce events, but only time-stamped hits. These data should be labelled with different items.

- **Sync pulses.** These data correspond to the AGATA clock counter time-stamped with the TDR clock counter. This data has to be used to calculate the phase clock difference  $\Delta T = TDR\ clock - AGATA\ clock$ . Other TDR data (three following paragraphs) should be re-time stamped using  $\Delta T$ . The consistency of  $\Delta T$  during the experiment should be checked by the merge (see paragraph 5). *Question: what to do if  $\Delta$  changes? One should define errors and associated actions.* Such data are expected each 655  $\mu$ s.

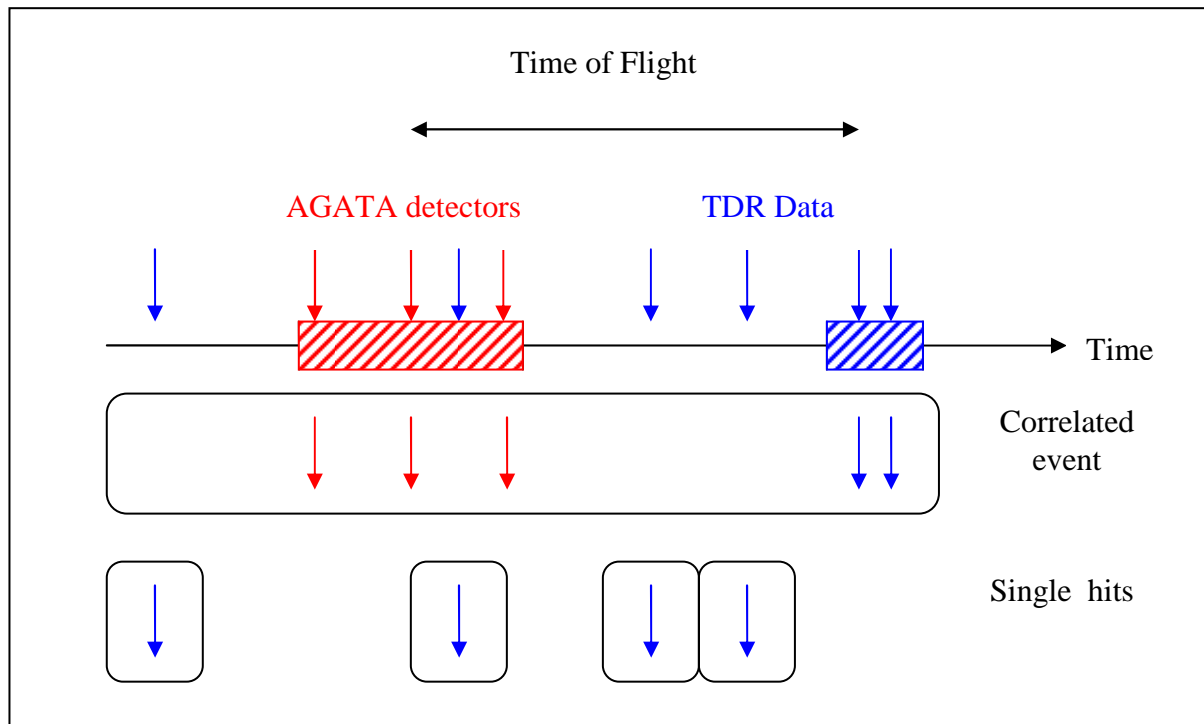


Figure 1 : merge of TDR data participating to the AGATA Trigger

- **Detectors participating to the trigger:** i.e. Silicon DSSD, Gas detector. The global event has to be reconstructed according to the trigger condition. Data not fulfilling the conditions have to be written to the data flow as single events. An example is shown in Figure 2: TDR correlated data are inside a time window (defined in the GTS) after the AGATA gamma time window. Search for such TDR data is performed looking forward from the AGATA time window. Other TDR hits corresponds for instance to radioactive decay of nuclei implanted at the focal plane.  
The hit rate expected for such event is  $\sim 1$  kHz.
- **Data not participating to the trigger:** i.e. Germanium focal plane detector, pin-diodes... Usually, the data are written to the data flow as single hits without further reconstruction, although some correlations may exist between these data. Including a more complex event reconstruction would significantly complicate the merge task; *to be discussed with JYFL*.  
The hit rate expected for such event is a few kHz.
- **Prompt gamma ray detectors,** i.e. Jurogam. These detectors should be merged using the same time window as AGATA. An example is shown Figure 2. Note that the TDR event-builder is already filtering the prompt gamma-ray flow using a similar technique.  
Received hit rate after filtering in the TDR is a few kHz.

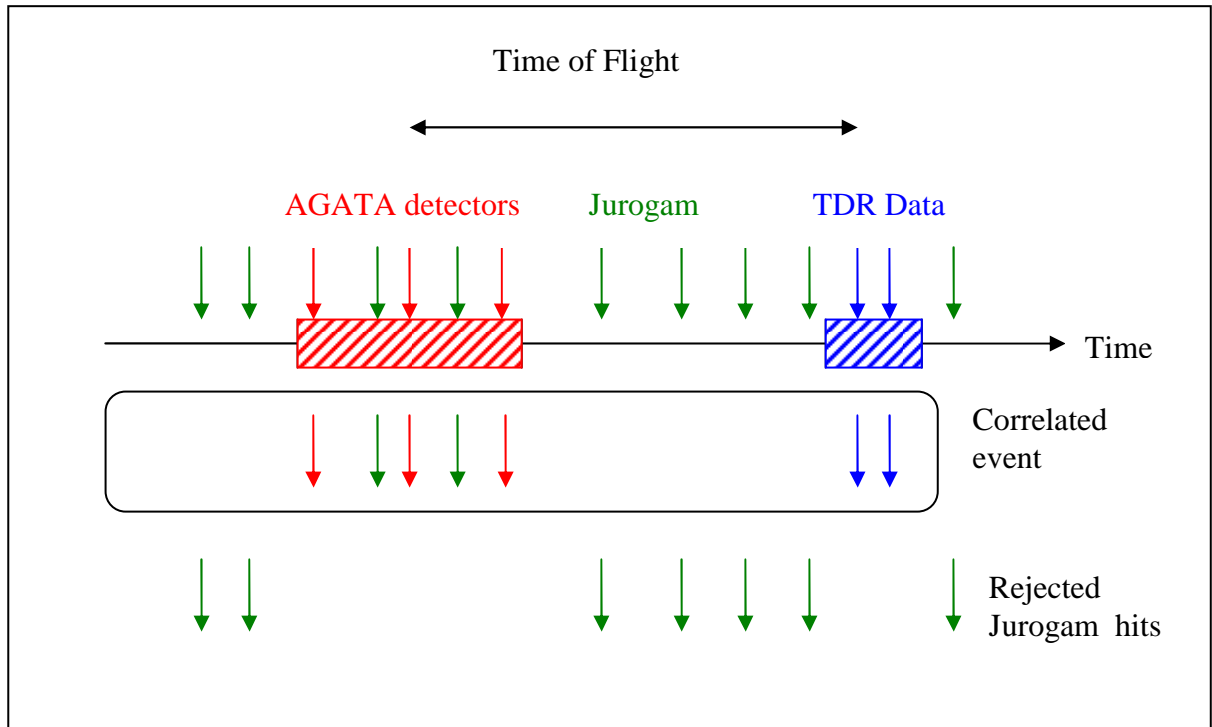


Figure 2 : merge of Jurogam data.

#### 4. Merge architecture and exchange protocol

The proposed architecture of the data acquisition is shown in Figure 3. The ancillary detector merge will be performed after the AGATA merge. The merge of ancillary detectors has to be performed before the tracking farm since they should provide the velocity vector used in tracking algorithm and for Doppler correction. Each ancillary detector DAQ will send his data to a NARVAL producer. The data exchange protocols for in different contexts are described in the next paragraphs.

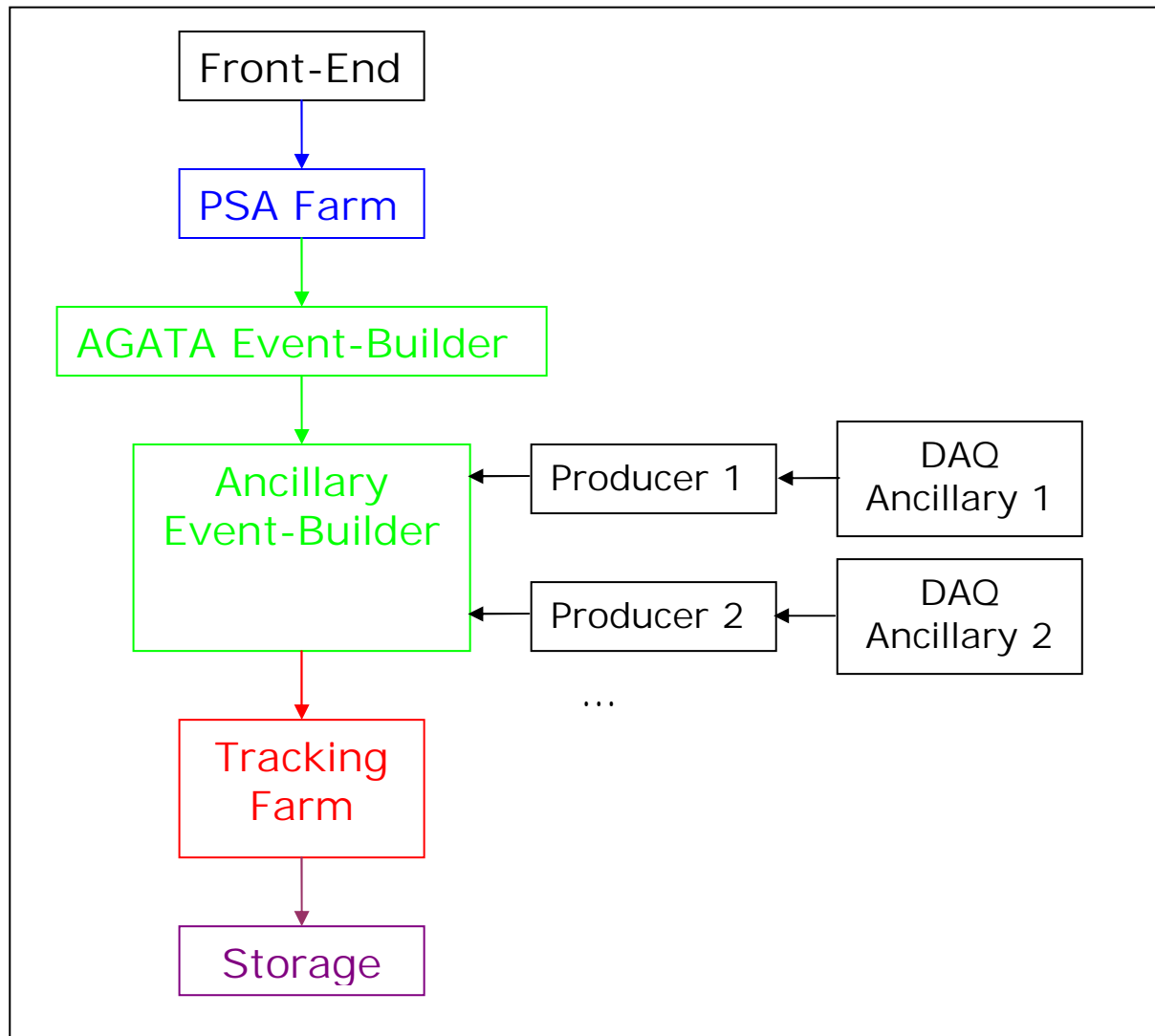
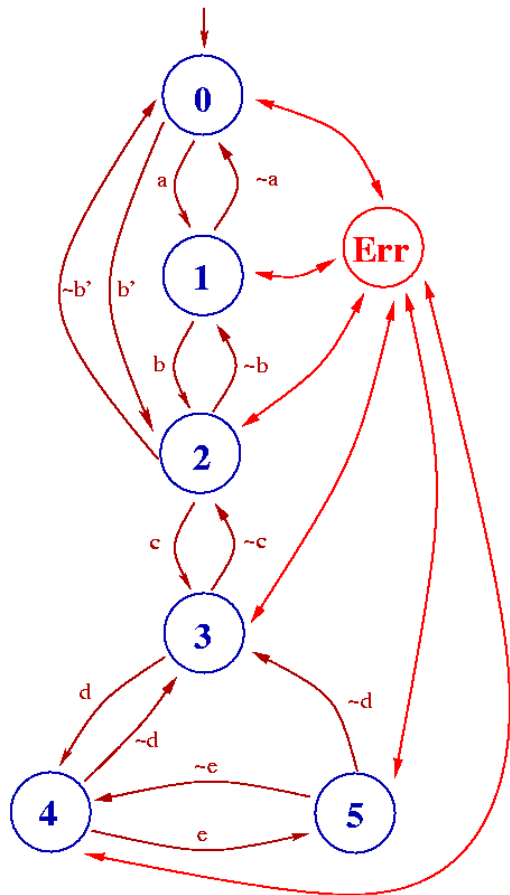


Figure 3: Data acquisition architecture

The ancillary detector data acquisition has to be compatible with the NARVAL automats states. Basic examples are actions associated with *reset*, *start*, and *suspend* transitions.

Although a full understanding of NARVAL states and transitions is not required, basic knowledge is summarized here. The description of NARVAL automats, taken from the NARVAL documentation, is presented Figure 4.

*NARVAL's automats***Figure 4 : NARVAL's automats.**

Description of the states:

- 0: *Initial state*. Nothing is configured and nothing is initialized.
- 1: *Configured state*. Nothing is charged, but it knows what to do.
- 2: *Charged state*. All of the actors are charged, but network, DSP ... aren't initialized.
- 3: *Ready state*. Everything is OK, network, DSP ..., are initialized.
- 4: *Running state*. The acquisition program is running...
- 5: *Suspend state*. The run is suspended.
- Err: *Error state*. When an error occurs somewhere, NARVAL goes in this intermediary state to decide what to do.

Description of the transitions:

- a: *Configure NARVAL*. During this action, we define all the actors needed by this Narval instance.
- b: *Charge Narval actor*. During this action, we charge all the actors required by the configuration step.
- b': *Configure and charge Narval*. This action does the same thing as "action a then action b".
- c: *Initialize Narval*. During this action, we initialize

the different parts of Narval: network interconnection, DSP ...

- d: *Start*. This is the start action. As everything is already initialized, we just have a 0 bit to change.
- e: *Suspend state*.
- ~\*: All the transitions where the name begins by a '~' have the exact opposite effect of the transition without the '~'. For example, d means START, so ~d means STOP.
- Error: When an error occurs, instead of changing to a standard state, Narval changes to the Error state

## 4.1. Default protocol

It is suggested to implement this protocol when none is implemented yet.

**Question: do we want to keep this section, or are we happy with specific protocols (GANIL, MIDAS, ...) only ?**

### 4.1.1. Identification

- In a first step, a merge port number and IP address will be assigned to each ancillary detector for data exchange. The ancillary detector will ask this information to a web server. The ancillary DAQ will then use the TCP/IP protocol for data exchange.
- In a second step, the DAQ will sent to NARVAL identification fields as followed :
  - 1 byte: length N of next string.

- N bytes: identification string (*i.e.* name of the ancillary detector) without end of string byte.
- 1 byte: little(1)/big(2) endian flag
- A 32 bit word corresponding to data buffer size N in byte.

Vic and Fred, could you suggest error messages, and suggest a flag name for intel VME processors?

After the identification, the acquisition is in step 3 (Ready state, see Figure 4) *i.e.* the ancillary detector is ready to sent data. If a transition  $\sim c$  between 3 and 2 states occurs at this level, the identification has to be restarted from the beginning.

#### 4.1.2. Buffers

For each, the DAQ will send:

- ~~A 32 bits word corresponding to the buffer size N in bytes. N equal 0 means end of transmission. This should be used for instance after DAQ stop (corresponding to a  $\sim d$  transition: see Figure 4).~~ Since this part has been removed, how do we send an end of data message? You suggested a heartbeat buffer: could you give details?
- A N byte buffer.

Could you suggest an error handling/recovery procedure?

The ancillary detector DAQ flush period should be compatible with the buffer merge depth. A maximum period of 1 buffer/s is recommended. Events should be time ordered.

## 4.2. Ganil Protocol

See appendix A.

## 4.3. Midas Protocol

## 5. Buffer decoding

The merge has to know what is inside the buffers and what to do with events or hits. The ancillary DAQ staff has therefore to provide a C (or even better ADA) library for this purpose.

An implementation of this library is proposed as followed:

- `int service(*buffer);` returns 0 is the event corresponds to detector data, 1 if correspond to a service message. A service message is for instance the TDR synchronization clock.
- `int process_service(*buffer);` service message process. In the case of the TDR, this function will calculate  $\Delta T = TDR\ clock - AGATA\ clock$ .
- `int event_size(*buffer);` returns the event size in bytes.
- `int event_number(*buffer);` returns GTS event number.
- `long long int clock_AGATA(*buffer);` returns the event/hit AGATA clock counter. In the case of the TDR, this function will calculate this counter using  $\Delta T$  and the TDR clock counter.
- `int event_type(*buffer);` return the event type, corresponding to different processing. This is for instance the case with the TDR acquisition where different windows have to be used whether the current event corresponds to the gas detector, Implantation Si detector,...

- `int keep(event_type);` return 1 if the detector channels has to be keep in the data flow even if the trigger or merge conditions are not fulfilled. Examples for the TDR are given in paragraph 3.2.

Parameters such as coincidence windows widths and delays will be parameterised through a web server. There will be as many parameters as event types.

*Error messages should be defined.*

## **6. Future ancillary detectors: upgrades to parallel readout**

It is expected (or at least recommended) that future ancillary detectors will be time-stamped using the AGATA clock counter. Event will be reconstructed from hits according to the GTS trigger conditions, as shown in the paragraph 3.2. Data rate may be as high as AGATA detector rate. More computing power could be required for future ancillary detectors using a full parallel readout, but estimates are difficult to make nowadays.

## 7. Interaction with the run control

We have to make clear that the ancillary detector should use its own slow control and GUI for the setup of the apparatus.

However, a minimum of interactions with the run control have to be defined for basic actions like *identification, setup, start, stop, suspend, initialise, reset*. The run control should also be able to get the status of the ancillary detector DAQ : *ready, start, stop....*

Architecture of the RUN control proposed by G. Maron (AGATA week Nov. 2005) is shown in Figure 5. The different components of the system, including ancillary detectors, are interfaced through a Function Manager (F.M.) using web services.

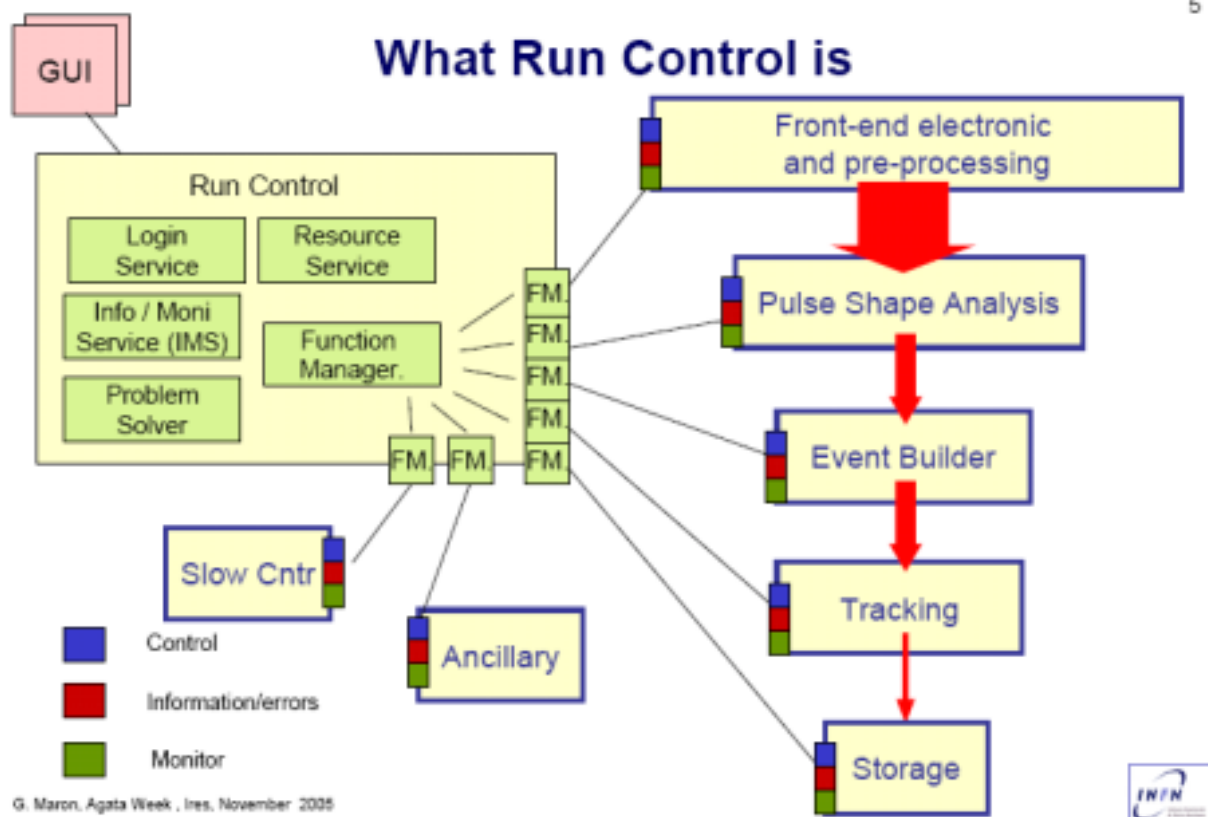


Figure 5: Run control (G. Maron).

## **8. Interaction with tracking data analysis**

In some cases, the ancillary detector will provide the velocity vector and the beam position on the target (beam tracker). It is not expected to improve the tracking efficiency using these information. However, it has been shown that the spectrum quality depends strongly on the velocity vector accuracy. Without a proper correction, spectra built in the tracking farm may be of very bad quality and therefore useless.

Providing the velocity vector at this level means that the ancillary detector should be fully analysed. This is not realistic since the right place is the Data Analysis farm. Therefore, the ancillary detector will NOT provide the velocity vector to the tracking farm. Further tracking improvement using the ancillary detector may be performed offline if necessary. This requires the storage of all AGATA interaction points.

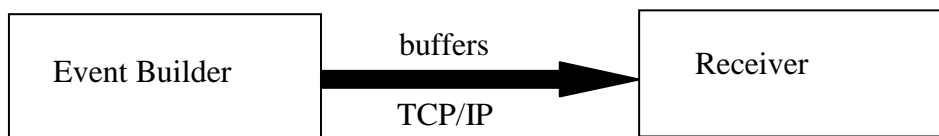
The Data Analysis framework, in the general context of Agata is being discussed. Ancillary detectors requests should be collected for a better definition of the Data Analysis tools, in close collaboration with the Data Analysis team.

## A GANIL Data transfer protocol

*Written by F. Saillant; Version 1.0 – December 7, 2005.*

This appendix describes the protocol used at GANIL to transfer the data buffers onto the network. Its goal is to help the developer to write a receiver program, hence we will take the receiver's point of view.

The GANIL protocol relies on TCP/IP and the standard BSD socket interface. The receiver is a TCP/IP client which receives the data buffers from a TCP/IP server – the “Event Builder”.



The receiver program must implement 3 procedures: an initialization procedure, a buffer reception procedure and a close connection procedure.

### A.1 Initialization procedure

This phase consists of 3 commands sent to the event builder in order to initialize the connection used to transfer the data buffers. These are the “GetAddress” command, the “SetBufferSize” command and the “StartTransfer” command.

### A.2 GetAddress command

The first command is intended to get the address of the TCP/IP server that will send the data buffers. It is not necessarily the same as the server handling this command.

The receiver must open a socket on port 3000 of the event builder, then send the GetAddress request, wait for the answer, send a “Close” request, wait for the acknowledgement and finally close the socket.

Format of GetAddress request: 24 bytes

```

req[0] = 0x01
req[1] = 0x9E
req[2] = 0xC6
req[3] = 0xE2
req[4..14] = 0
req[15] = 0x10
req[16..23] = 0
  
```

Format of the answer to the GetAddress request: 104 bytes

`answer[0..23]` is an acknowledgement – do not care  
`answer[24..103]` is an ASCII character string containing the IP address followed by a protocol version number (separated by a white space). Currently the version number is 4.

Format of Close request: 24 bytes

```
req[0] = 0x01
req[1] = 0x9E
req[2] = 0xC6
req[3] = 0xE2
req[4..14] = 0
req[15] = 0x02
req[16..23] = 0
```

Acknowledgement of Close request: 24 bytes - do not care

### A.3 SetBufferSize command

This command allows setting the buffer size. So far the size must be 16384 bytes. A socket must be opened on port 3001 of the address provided by the GetAddress command.

Format of SetBufferSize request: 12 bytes

```
req[0..2] = 0
req[3] = 1
req[4] = 4
req[5..11] = 0
```

The size is coded on `req[4..7]`. The value is thus 0x4000. It is the only value used in GANIL DAQ so far.

Format of the acknowledgement: 12 bytes

```
ack[0..2] = 0
ack[3] = 0 if OK, 1 if ERROR
ack[4..11] = 0
```

### A.4 StartTransfer command

This command must be sent using the same socket as the previous one. So the socket must not be closed between these two commands. The StartTransfer command asks the event builder to send data buffers continuously as they are produced.

Format of StartTransfer request: 12 bytes

```
req[0..2] = 0
req[3] = 2
```

```
req[4..11] = 0
```

The StartTransfer request has no specific acknowledgement (each data buffer is actually an acknowledgement).

## A.5 Buffer reception

After sending StartTransfer, the receiver must read the data buffers from the socket. The 12 byte tail of the buffers is an acknowledgement with the same format as the acknowledgement of SetBufferSize. A third value for ack[3] (ack[3] = 2) is used to indicate that the buffer contains no data, i.e. no data buffer was produced by the DAQ.

## A.6 Close connection

To close the connection the receiver must open a new socket on port 3001 of the address provided by GetAddress, then send a “CloseConnection” request, wait for its acknowledgement and finally close both sockets.

Format of CloseConnection request: 12 bytes

```
req[0..1] = 0
req[2] = 0xFF
req[3] = 0xFF
req[4..11] = 0
```

Acknowledgement: 12 bytes – do not care.