Development of DIGI access and data formatting code in ORCA for DAQ applications and test beams

Giacomo BRUNO

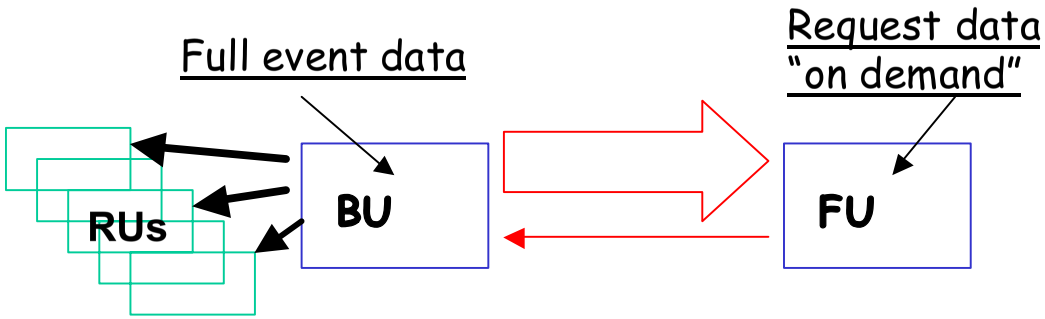
CERN-EP Division

RPROM/SPROM joint meeting 24 February 2003 (CMS week)

- the DaqPrototype for on-line and test beams
- Digi access structure in COBRA/ORCA
- Tracker and Muon digi access implementations
- Proposal for new “daq” tracker and muon specific code
- Conclusions

On-line event selection

The ORCA reconstruction software has to be used by the DAQ to accomplish the HLT selection on-line. To this aim it is necessary to integrate ORCA in the DAQ code (XDAQ).



Raw data is received, upon request, by the FU. The BU sends it in fragments corresponding to subsets of detectors, served by Readout Units.

RU1 serves Det_1_1

Det_1_2

....

RU2 serves Det_2_1

Det_2_2

On-line event selection (2)

Here it runs a XDAQ executive.

It provides:

- Steering code handling communication with other Daq components (FU in particular)
- Monitoring
- ….

One must add:

- Input formatted event data

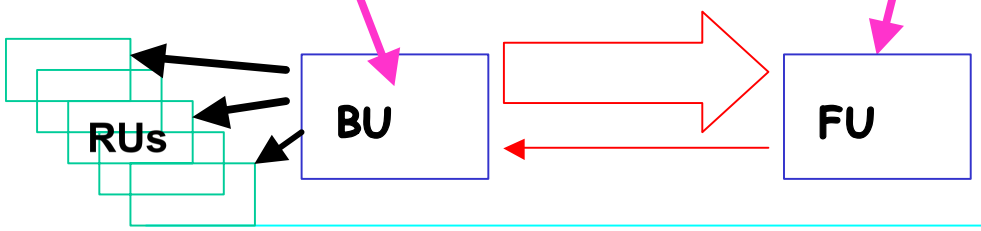
Here it runs a XDAQ executive.

It provides:

- Steering code handling communication with other Daq components (BU in particular)
- Monitoring
- ….

One must add:

- HLT selection algorithm based on the ORCA reconstruction code
- Data formatting/unformatting
- Selective Persistency



What is needed?

- Creation of raw data: transform simulated detector data (OO digi on a DB) into raw data arbitrarily formatted and write it on a certain support (binary file).
- Provide a tool to read in and allocate this raw data (BU task)
- Provide a “hook” to ORCA such that an HLT selection algorithm can run in the FU XDAQ executable.
- Provide a tool to interpret the raw data allowing the ORCA reconstruction code to use it (transform raw data into OO digis and fill the DetUnits digi cache).

All this can be achieved by designing the code in a way such as to provide an additional functionality:

- to run an ORCA stand-alone reconstruction application in which the input data is raw data (under an arbitrary format) fed through a simple binary file. Very interesting for test beam off-line data analysis (or even on-line).

Code development in COBRA/ORCA

- Subdetector independent part (in COBRA)

  A new COBRA subsystem, the DaqPrototype has been developed and is already present in the COBRA_7_1_2 release.

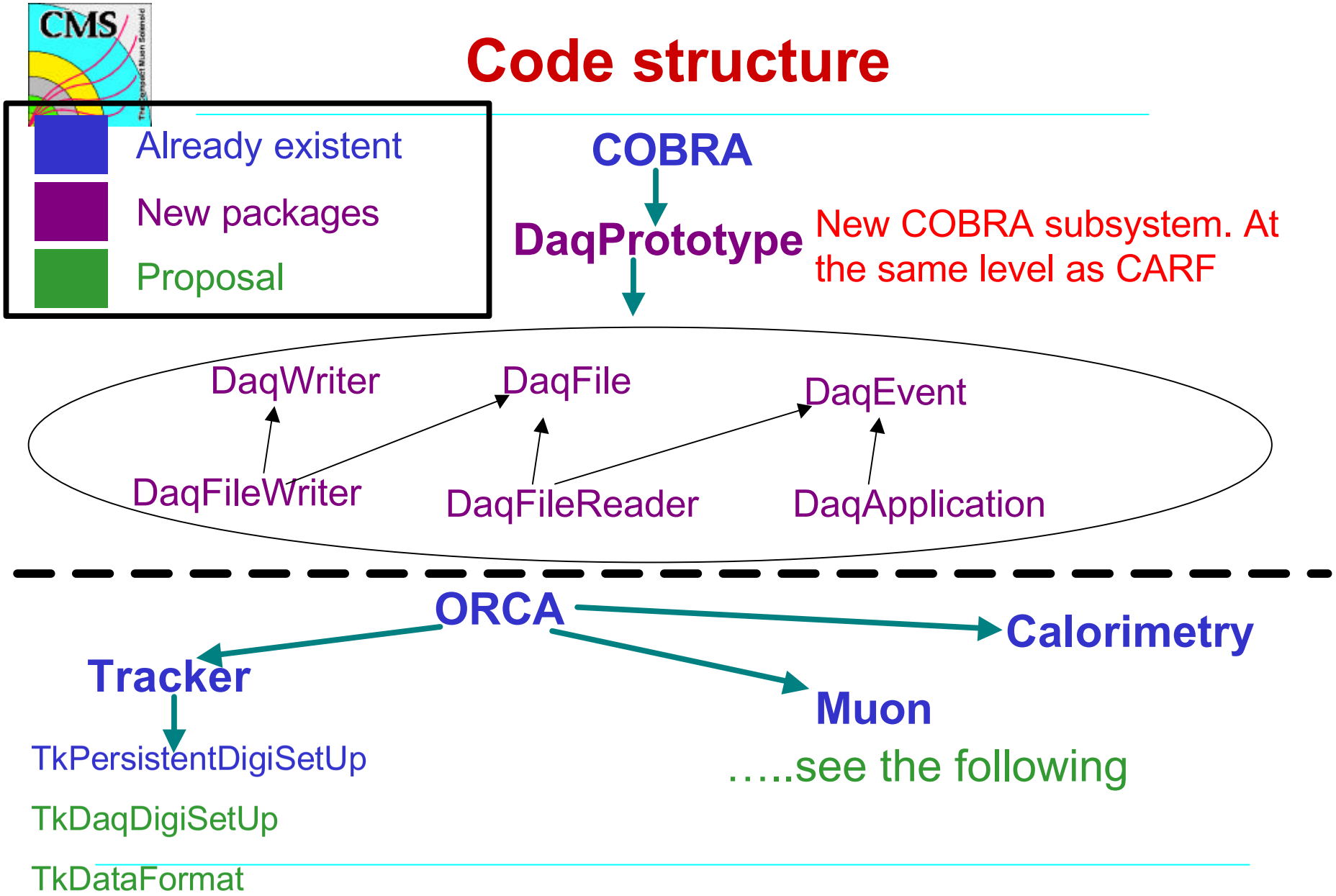
  – XDAQ – ORCA interface
  – Raw data allocation
  – Simple Persistency

- Sub-detector dependent part (in ORCA)

  DaqPrototype must be complemented by a few detector specific packages, not yet in release.

  – Raw data – DetUnits interface
  – Raw data formatting/unformatting

Code structure



The executable for the BU

The (XDAQ) executable running on the BU needs to link the following DaqPrototype sub-Packages.

- DaqEvent → Raw data allocation
- DaqFileReader
- DaqFile

} Reader

At the moment, raw data is stored on a binary file and read in from it.

The executable for the FU

The (XDAQ) executable running on the FU needs to link the following DaqPrototype sub-Packages in addition to the standard ORCA ones and some COBRA ones

- DaqApplication → Interface to ORCA reconstruction packages
- DaqEvent → Raw Data , basic formatting/unformatting
- DaqWriter
- DaqFileWriter
- DaqFile

} Persistency

the standard COBRA persistency mechanism was not made to work with OBJECTIVITY, not yet tried with ROOT.

- Some new detector-specific packages (e.g. for the Tracker):
  - TkDaqDigiSetUp — Raw data – DetUnits interface
  - TkDataFormat — Final data formatting/unformatting

How to get the raw data file?

One simply runs a standard ORCA Sim or Rec application to which he links, in addition, the following DaqPrototype sub-Packages (just correct BuildFile):

- DaqEvent → Raw Data , basic formatting/unformatting
- DaqWriter
- DaqFileWriter
- DaqFile

} Persistency on a binary file

- Just one of the new detector-specific packages (e.g. for the Tracker):
  - ➢ TkDataFormat — Final data formatting/unformatting

And analyzing test beam data…

One can also simply run a generic ORCA reconstruction analysis having some input raw data (under any format) on a binary file (instead of the DB) by just correcting his BuildFile and link, instead of the Sim/RecReader groups,:

- DaqApplication → in this case a new ORCA application
- DaqEvent → Raw Data , basic formatting/unformatting
- DaqWriter
- DaqFileWriter

  } Simple Persistency (if desired)
- DaqFile
- DaqFileReader

  } Raw data reader
- The new detector-specific packages (e.g. for the Tracker):
  - TkDaqDigiSetUp — Raw data – DetUnits interface
  - TkDataFormat — Final data formatting/unformatting

Data Access/Formatting

Detector raw data will be delivered on demand in chunks corresponding to groups of physical detectors.

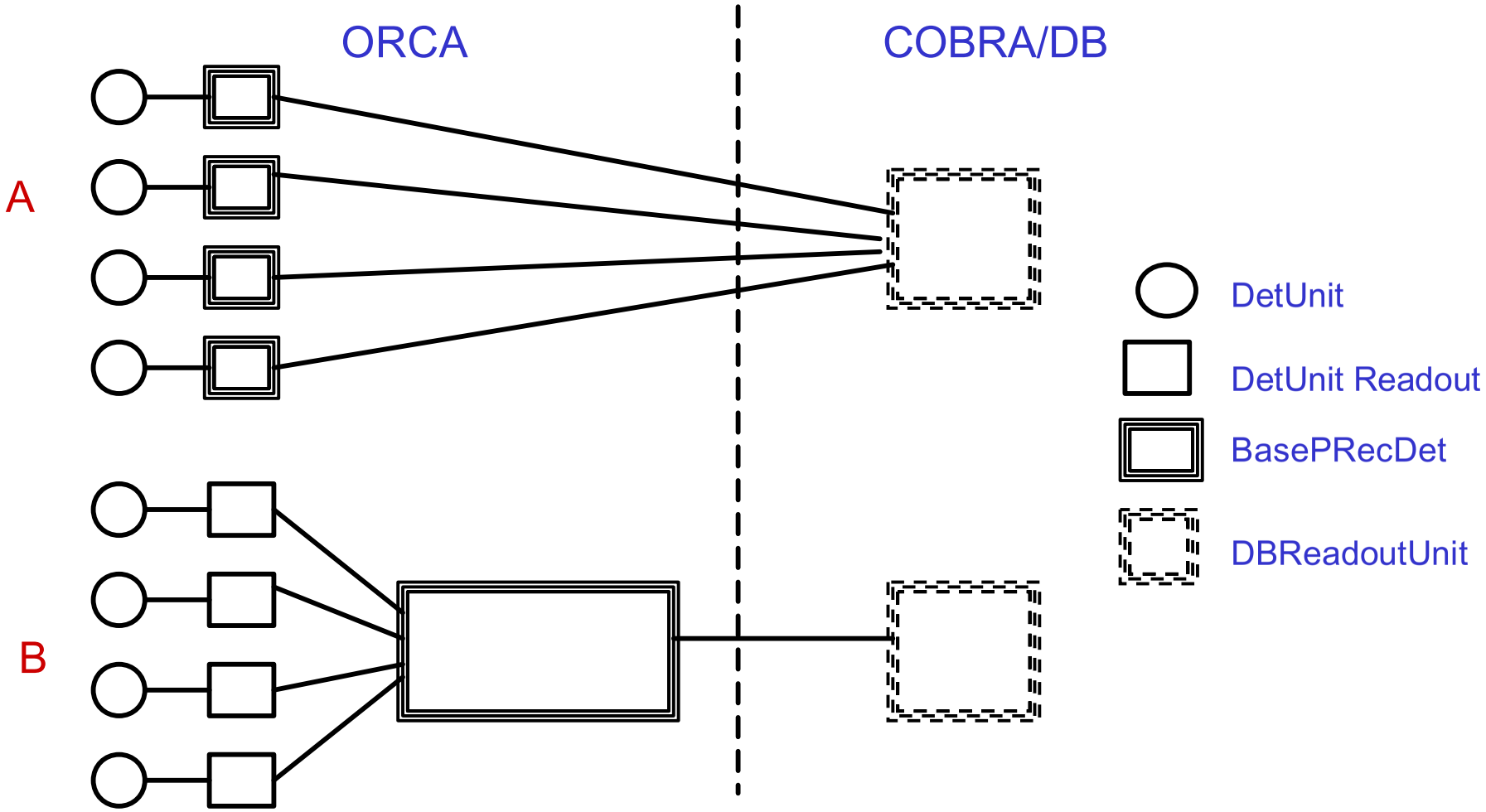
Therefore, a Master (Readout Unit) – slaves (DetUnits) digi access structure should be reproduced in ORCA. The reason for this is two-fold:

1. Facilitate handling and interpretation of raw data.
   unformat raw data, construct OO Digi, fill up the DetUnit caches.
2. Optimization of reconstruction algorithms for the on-line HLT event selection.
   the algorithms should take into account how many bunches of data on average need to be requested to come to a conclusion (e.g. the Tracker will group its detectors in wedges rather than layers, so that track regional reconstruction in a cone will most probably require only one raw data request)

Digi access in ORCA: general idea

1. Each sub-detector registers a *DB ReadoutUnitFactory* capable to create *DB ReadoutUnits*. The *DB ReadoutUnit* knows how to access the corresponding digi data on the DB and is reachable through a name (*SuId*).
2. On dispatch of the SetUp, all *DetUnits* are constructed and each of them is given its own *Readout* (slave). There are two possibilities:
   A. The *DetUnit Readout* is (or owns) a *BasicPRecDet*, which is a *LazyObserver* of *G3EventProxy*. The *Readout* is constructed with a master RU name (*SuId*). If a new name is given, the *DBReadoutUnitFactory* will create a new *DB ReadoutUnit* object.
   B. A number of master RUs (*BasePRecDet*) are constructed, so they are the *lazy observers of G3EventProxy*. They are constructed with a name (*SuId*), so the *DBReadoutUnitFactory* will construct as many *DB ReadoutUnits*. The *DetUnit Readouts* do not need to be *BasePRecDets, but* are simply "connected" to these masters

Digi access structure



Possible solutions

First solution: act on the DBReadoutUnits.

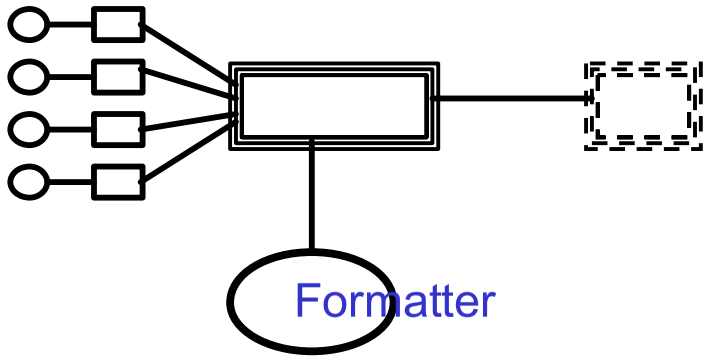
However, COBRA has been designed to run reconstruction applications that read from and write on the same support (an OO DB). To achieve the goal of reading from an external source, run a reconstruction application and write on a DB would require modifications in several COBRA packages.

Second solution: act on the BasePrecDets.

It is just needed to route the request for data to something other than the DB ReadoutUnits. It would be desirable not to change the channel for writing out (Persistency mechanism). In the case of scenario A there is a drawback: one needs to insert a Master RU interface between the DetUnits and the raw data.

The Tracker choice: B

The Tracker package TkPersistentDigiSetUp contains an Observer of the SetUp. It triggers the geometry/DetUnit construction, then the master RUs are constructed and the DetUnits are connected to them. No other package depends on TkPersistentDigiSetUp.



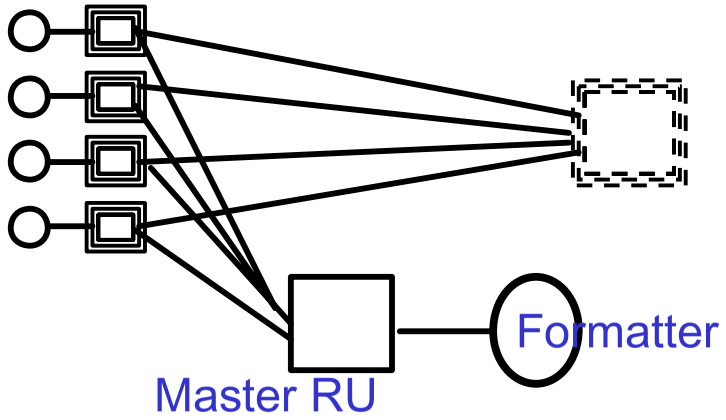
What has been added:

1)When one reads data from something else than a DB, he will replace TkPersistentDigiSetUp with a new package Tracker/TkDaqDigiSetUp. The possibility to write out on the DB is always there and could be restored at any time.

2)The master RU is supported by a Formatter, which has the task to interpret the raw data of the RU and fill the DetUnit caches (and vice versa, to create raw data): new package Tracker/TkDataFormat

The Muon choice: A, well…

All three Muon sub-detectors have a package called MXXXSetUp. It contains an Observer of the SetUp that constructs the full geometry, attaches to each DetUnit its readout (BasePRecDet) and gives to it the name of the Master RU. Several other muon packages depend on MXXXSetUp.



What needs to be added:

1) Two new interchangeable packages: MXXXPersistentDigiSetUp (when reading Digi from a DB) and MXXXDaqDigiSetUp (when reading raw data from a different source) implementing the RU Master interface.

2) A new package Muon/MXXXDataFormat providing the master RU with a Formatter, which has the task to interpret the raw data of the RU and fill the DetUnit caches (and vice versa, to create raw data).

Actual digi access implementation(1)

DT:

• the DetUnit Readout has a private class that is a PRecDet, which is a BasePRecDet. MuBarDigi information is compacted in one single integer as it is done in the Tracker

CSC:

• the DetUnit (one of the 6 CSC Layers) has one Readout which is a LazyObserver of G3EventProxy. The Readout also owns two SlaveRecDet (Wire and Strip Digi), that are connected to two MasterRecDet, which is a BasePRecDet. (two LazyObservers of G3EventProxy involved!). MuEndWireDigi

RPC:

• The DetUnit Readout is a RecDet that is a LazyObserver of G3EventProxy. For Digi access and persistency, another class is involved: MRpcGlobalReader that is a PRecDet

Actual digi access implementation(2)

In all cases there is inefficient digi access interface: digis are accessed through copies of objects of type vector<MuXXXDigi> instead of Ranges of iterators as it is done in the Tracker.

MuBarDigi: unsigned int theData: compacted in one single integer as it is done in the Tracker

MuEndWireDigi: int theWireGroup; int the5thElectronTime; int theFDTime; int theBeamCrossingTag; int theAdcCounts;

MuEndStripDigi:int theStripNumber; int theComparatorOutput; int theComparatorTime; int theBeamCrossingTag; int theAdcCounts; int theScaCounts[N_SCA_BINS]; float theScaStartTime; N_SCA_BINS=8/16 it can make up to 100 bytes per digi! 25 times more than one Tracker digi. A fast and perhaps wrong computation: one digitized CSC event with neutron background is as large as the Tracker digi event at high luminosity!?

MRpcDigi:int ChamberId, int StripId, int BxId;

My personal DT Digi access implementation

"Minimum change strategy" .The DetUnits keeps on having a Readout which is a BasePRecDet. Addition of the master RU interface:

- Muon/MBSetup (old): Two new abstract classes and addition of one line in an existing class
- Muon/MBDetector (old): Two existing classes modified (MuBarChamber, MuBarBaseReadout)
- Muon/MBDaqDigiSetUp (new): to read from Daq sources.
- Muon/MBPersistentDigiSetUp (new): to read Digi from a standard DB (implementation of the abstract classes in MBSetup).
- Muon/MBDataFormat (new): format/unformat

My personal CSC Digi access implementation

"Minimum change strategy" . The DetUnits keeps on having a Readout which is no longer a LazyObserver of G3EventProxy, large simplification. Extension of the already present master RU interface:

- Muon/MESetup (old): Two new abstract classes and addition of one line in an existing class
- Muon/MEDetector (old): Two existing classes modified (MuEndPRecDet, MuEndReadout)
- Muon/MEDaqDigiSetUp (new): to read from Daq sources.
- Muon/MEPersistentDigiSetUp (new): to read Digi from a standard DB (implementation of the abstract classes in MESetup).
- Muon/MEDataFormat (new): format/unformat

My personal RPC Digi access implementation

I tried to port to scheme B used by the Tracker. The DetUnits now have a Readout which is no longer a LazyObserver of G3EventProxy. The already present “pseudo” master RU interface has been completely revisited. Efficient digi access interface. Large use of CommonDet code:

- Muon/MRpcSetup (old): Two new abstract classes and addition of one line in an existing class
- Muon/MRpcDetector (old): Two existing classes modified (MRpcReadout, MRpcDetUnit)
- Muon/MRpcDaqDigiSetUp (new): to read from Daq sources.
- Muon/MRpcPersistentDigiSetUp (new): to read Digi from a standard DB (implementation of the abstract classes in MESetup).
- Muon/MRpcDataFormat (new): format/unformat

Conclusions (1)

- A new COBRA subsystem has been developed and is present in the present COBRA_7_1_2 release: COBRA/DaqPrototype.
  - It allows to run a prototype of the DAQ Builder Unit-Filter Unit system, the FU being able to run any HLT selection algorithm based on the ORCA reconstruction code. Raw data can be fed into the BU under any format, Raw data will be interpreted in the FU. Interesting for on-line test-beam analysis.
  - In the XDAQ environment the COBRA persistency mechanism was not made to work (not yet tried with ROOT), so it is disabled. An alternative, simple persistency mechanism has been provided: write-out on a binary file.
  - A "spin-off" functionality exists: the code has been designed so that one can also run a generic ORCA reconstruction analysis having some input raw data (under any format) on a binary file (instead of the DB). Interesting for off-line test-beam analysis.

Conclusions (2)

- The DaqPrototype must be complemented by a few new detector-specific packages (going into ORCA, not yet in release) providing the following functionalities: 1)Raw data – DetUnits interface 2)Raw data formatting/unformatting
  - For the Tracker a tested implementation already exists: (two new packages TkDaqDigiSetUp and TkDataFormat). It Makes large use of CommonDet code. The implementation is very clean thanks to the nice underlying digi access code structure.
  - The Muon digi access code is less suitable for a straightforward clean implementation. The various sub-detectors have followed different paths, which are not the most efficient. A preliminary, not fully tested version of the code has been implemented. The Rpc version conforms to the Tracker one. It could be done the same for DT and CSC with the enormous advantage brought by uniformity among all detectors and use of CommonDet code.