HGCAL for ILD @ LLR - February 2015
Linear Collider Data Quality Monitoring (LCDQM)

Eté Rémi <rete@ipnl.in2p3.fr>

Université Claude Bernard Lyon 1 - Institut de Physique Nucléaire de Lyon

2 février 2015

LCDQM : a monitoring system

Why a common monitoring system for our data ?

LCDQM : a monitoring system

Why a common monitoring system for our data ?

$\rightarrow$ The answer is in the question !

LCDQM : a monitoring system

Why a common monitoring system for our data ?

$\rightarrow$ The answer is in the question !

1. a common tool to monitor our data

LCDQM : a monitoring system

Why a common monitoring system for our data ?

$\rightarrow$ The answer is in the question !

1. a common tool to monitor our data
2. important need for common test-beams (ECAL + HCAL)

LCDQM : a monitoring system

Why a common monitoring system for our data ?

$\rightarrow$ The answer is in the question !

1. a common tool to monitor our data
2. important need for common test-beams (ECAL + HCAL)
3. global view of what is going on during test beam

All collider experiments have their own DQM system (CMS, ATLAS, ALICE, LHCb, ...)

Many ideas here are taken from the ALICE DQM system, AMORE (B. Von Haller)

Global software overview

LCDQM : Linear Collider Data Quality Monitoring

Main ideas :

- Standalone run control for DQM
- Data distributing system (services) over the network
- Data processing adapted to DQM (+ archiving)
- Histograms distributing system (services) over the network
- Vizualization interface (GUI, web page)

Global software overview

LCDQM : Linear Collider Data Quality Monitoring

Main ideas :

- Standalone run control for DQM
- Data distributing system (services) over the network
- Data processing adapted to DQM (+ archiving)
- Histograms distributing system (services) over the network
- Vizualization interface (GUI, web page)

The software should consists mainly in executables ...

- start a run control
- start a standalone event service for lcio files
- start a module processing (see next slides)
- start a vizualisation interface (GUI or web page)

Global software overview

LCDQM : Linear Collider Data Quality Monitoring

Main ideas :

- Standalone run control for DQM
- Data distributing system (services) over the network
- Data processing adapted to DQM (+ archiving)
- Histograms distributing system (services) over the network
- Vizualization interface (GUI, web page)

The software should consists mainly in executables ...

- start a run control
- start a standalone event service for lcio files
- start a module processing (see next slides)
- start a vizualisation interface (GUI or web page)

... and libraries :

- to plug the data service and run control into the DAQ
- to plug the analysis modules that produce histograms
- to extend the vizualisation interfaces

Global software overview

LCDQM : Linear Collider Data Quality Monitoring

Main ideas :

- Standalone run control for DQM
- Data distributing system (services) over the network
- Data processing adapted to DQM (+ archiving)
- Histograms distributing system (services) over the network
- Vizualization interface (GUI, web page)

The software should consists mainly in executables ...

- start a run control
- start a standalone event service for lcio files
- start a module processing (see next slides)
- start a vizualisation interface (GUI or web page)

... and libraries :

- to plug the data service and run control into the DAQ
- to plug the analysis modules that produce histograms
- to extend the vizualisation interfaces

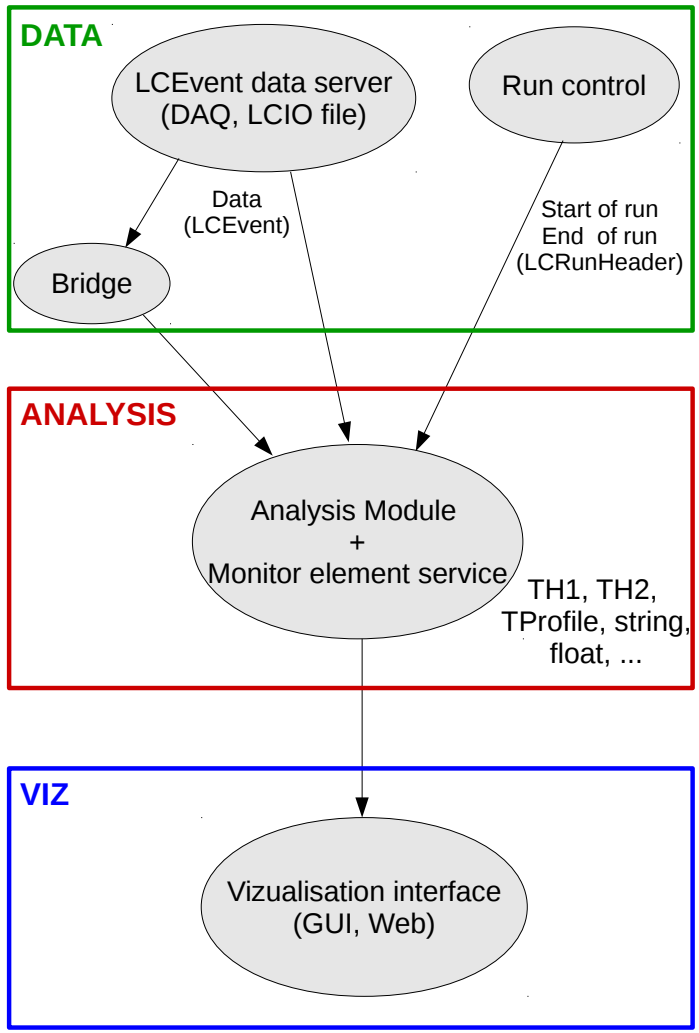
Dependencies : LCIO, ROOT, Qt (GUI), Wt (Web), DIM (networking). **All in C++ ! !**

Packages overview

Software mainly split in **three parts** :

- Data service : Publication of LCEvents (TCP/IP) via a service, standalone DQM Run Control (SOR, EOR), data serialization. Easy to couple to an DAQ system (XDAQ, EUDAQ)
- Data processing : Modules process data received by a service. Produce monitor elements (histograms, graphs, scalars, etc ...) and publish them via a service (TCP/IP).
- Vizualisation : Graphical/Web interface application receives monitor elements on request, organize them and display them.

Data and runs in LCIO format.
All services/client connections handled by DIM by TCP/IP (sockets).
Histograms, graphs, etc ... encapsulated in ROOT objects.
Vizualisation interfaces implemented with Qt (GUI) and Wt (Web).



Module application

Overview

A module is a user plug-in code like a Marlin processor. Its processes a fast analysis of the received data and produces monitor elements (histograms, graphs, scalars, ...). These elements are published in a service (TCP/IP) at the end of a cycle.

Monitor Element

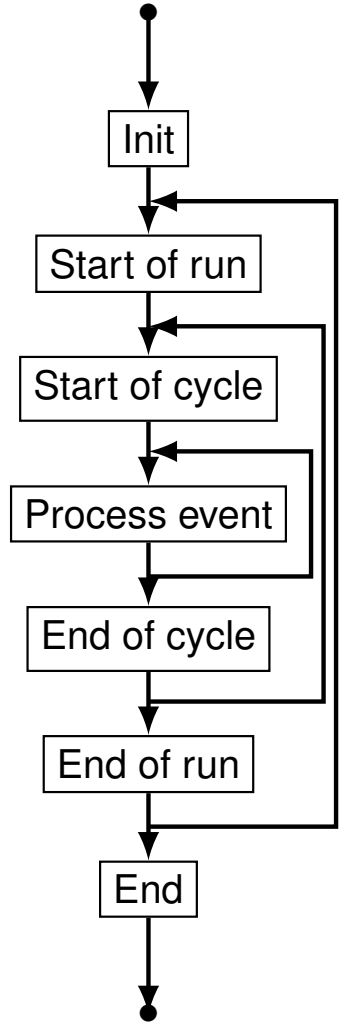
Wrapper of ROOT histograms, graphs, scalars, profiles, etc ... A monitor element is qualified by its quality, a name, a title, a reset policy (when the element has to be reset), a short description, a run number and its associated module name. Monitor elements are book via the DQMModuleApi (see next slides).

Module Cycle

Series of *processEvent(evt)* calls of a module. Can be of many types :

- Event counter cycle
- Timer cycle
- Event size cycle (N bytes)

At the end of a cycle, the list of monitor elements associated to its module are published via a service.



Module application flow

Module application

Module interface

```
class DQMModule
{
  virtual const std::string &getDetectorName() const = 0;
  virtual const DQMVersion &getVersion() const = 0;
  virtual StatusCode init() = 0;
  virtual StatusCode readSettings(const TiXmlHandle &xmlHandle) = 0;
  virtual StatusCode end() = 0;
  virtual StatusCode processEvent(EVENT::LCEvent *pLCEvent) = 0;
  virtual StatusCode startOfCycle() = 0;
  virtual StatusCode endOfCycle() = 0;
  virtual StatusCode startOfRun(EVENT::LCRunHeader *pRunHeader) = 0;
  virtual StatusCode endOfRun(EVENT::LCRunHeader *pRunHeader) = 0;
  virtual StatusCode reset() = 0;
};
```

```
class DQMModuleApi
{
static StatusCode bookRealHistogram1D(DQMModule *const pModule, DQMMonitorElement *&pMonitorElement,
    const std::string &name, const std::string &title, int nBins, float minimum, float maximum);
static StatusCode bookIntHistogram1D(DQMModule *const pModule, DQMMonitorElement *&pMonitorElement,
    const std::string &name, const std::string &title, int nBins, float minimum, float maximum);
static StatusCode bookCharHistogram1D(DQMModule *const pModule, DQMMonitorElement *&pMonitorElement,
    const std::string &name, const std::string &title, int nBins, float minimum, float maximum);
  // ...
};
```

Vizualisation interface

Gui implementation with Qt and web implementation with Wt (Qt-like for web).
Functionalities :

- module and monitor elements search
- histogram vizualisation
- comparison of histograms with **reference histograms**
- extensibility for a different viz implementation

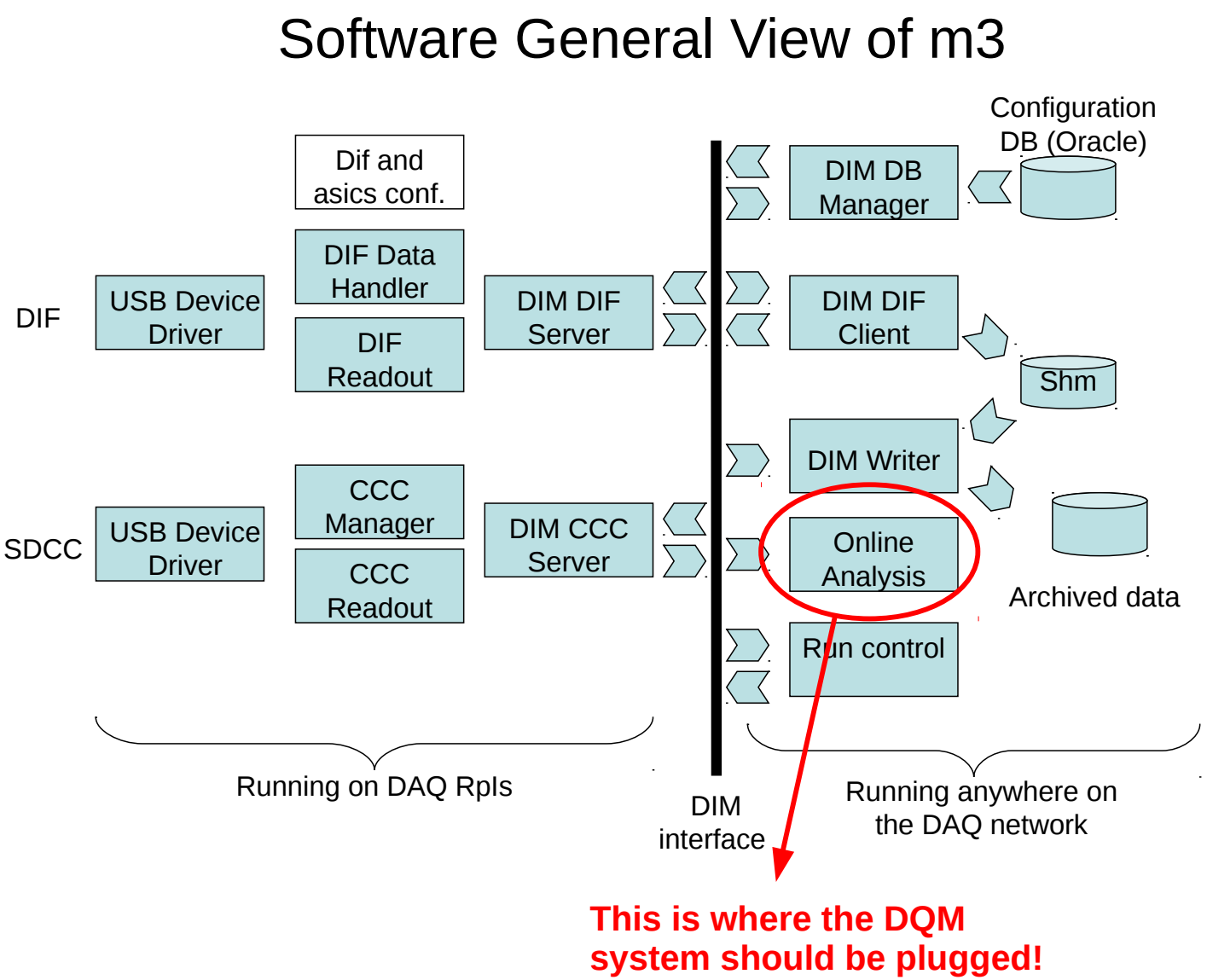
Qt implementation :

- advantage : root histogram handling, easy deployement (simple executable)
- drawback : need to install locally the package and open your viz

Web implementation :

- advantage : easy to handle -> open your browser and enter the correct address
- drawback : no root histogram handling (generated image), needs efforts on deployment

In sdhcal DAQ



Software status

Data serialization	DATA	OK	-
Event service	DATA	OK	-
Event client	DATA	OK	-
Run control	DATA	OK	-
Bridge service	DATA	NO	2 days
Memory check	DATA	NO	2 days
Module api	ANALYSIS	DEV	1 week
Module application flow	ANALYSIS	OK - DEV	1 week
Memory check	ANALYSIS	NO	3 days
Monitor element service/client	ANALYSIS	NO	2 weeks
Gui/Web implementation	VIZ	DEV	1 - 1.5 months

First working implementation forseen in april with SDHCAL modules implementation (V. Buridon)
Code available on Github : https ://github.com/rete/LCDQM.
Easy to install with cmake (needs DIM additionally)
For an ECAL implementation, please feel free to contact me to discuss on how to do it.
Possibility to have :

- SDHCAL modules
- ECAL modules
- Coupled ECAL+SDHCAL modules