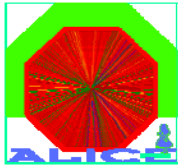


ALICE Online Data Quality Monitoring

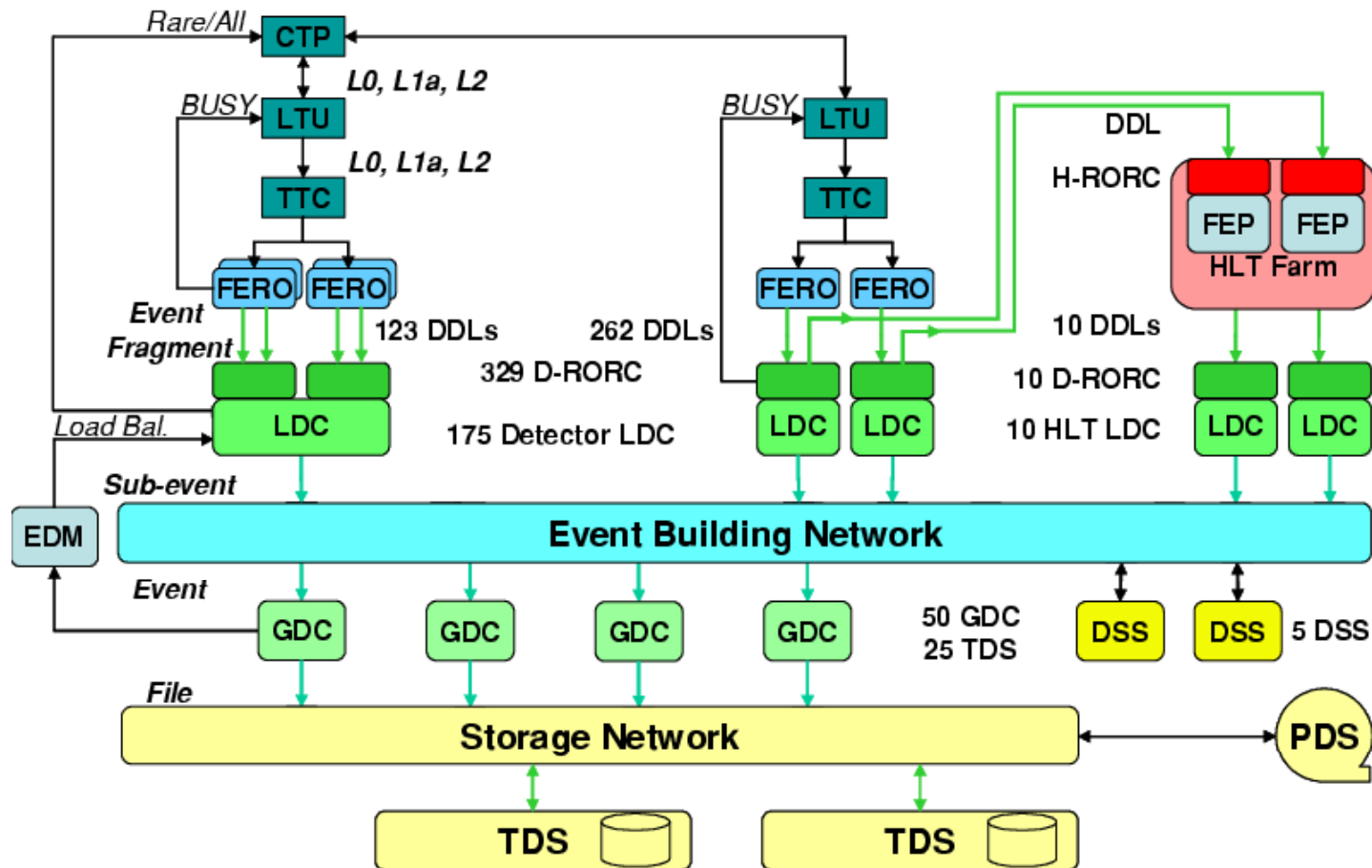
Filimon Roukoutakis

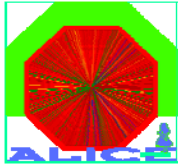
CERN PH-AID

ROOT Users Workshop
CERN, 27/03/2007



ALICE DAQ Architecture



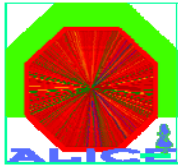


MOOD

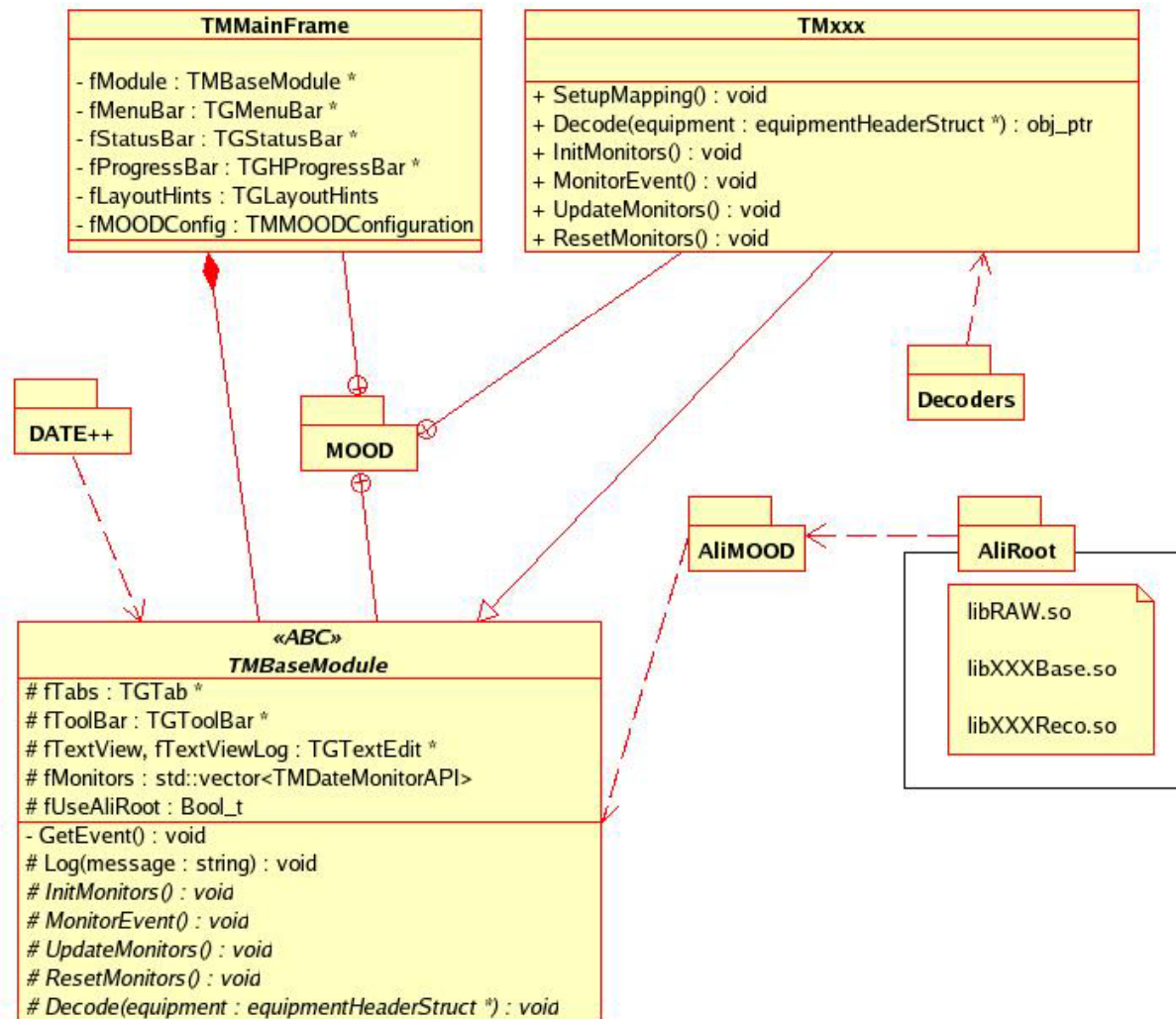
Monitor Of Online Data

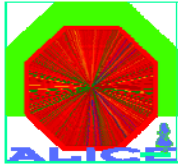


- Originally developed by Ozgur Cobanoglu for the DAQ group
- 2 Tasks: Online/Offline Data quality monitoring & Detector Debugging
- Written in C++, based on ROOT & DATE (ALICE DAQ Software)
- Runs under SLC 3/4, with DATE v5 and ROOT v5.12
- General framework has evolved, most of the core functionality (DATE interface, GUI) has been completely rewritten during summer 2006. The new architecture (next UML diagram) was introduced in release 5.00.01, 6 releases followed, now on 5.00.07.



MOOD UML Class diagram

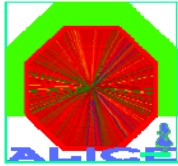




Summer 2006 changes



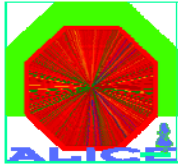
- All the core framework functionality (DATE interface, common GUI functionality) has been completely rewritten and is accessible through an abstract base module class. The module developer has to implement just 5 detector-specific pure virtual functions in each derived module.
- There is the possibility of creating a whole hierarchy of derived modules for each detector, sharing e.g. the decoding-mapping of the payload and differentiating in the visualization.
- The new framework is AliRoot-aware (AliROOT=the ALICE Offline framework), a detector module can readily utilize AliRoot classes for decoding, mapping. A first application of this is the ability to monitor files in the official ALICE offline (ROOT) format.
- Performance enhancements as a result of framework redesigning and multithreading. The slow updating of the screen view is now essentially detached from the continuous update of monitoring elements in the background.



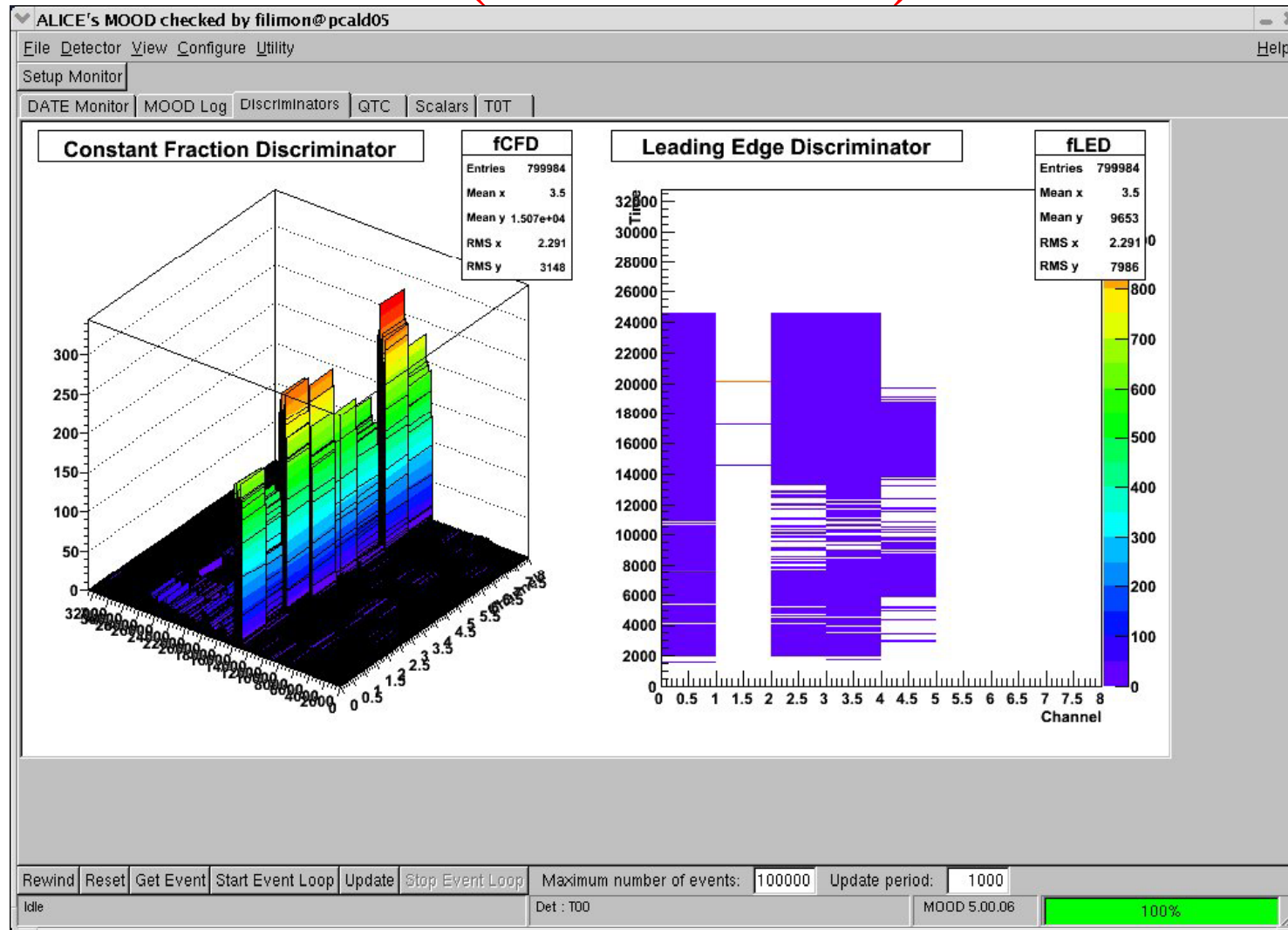
MOOD framework features

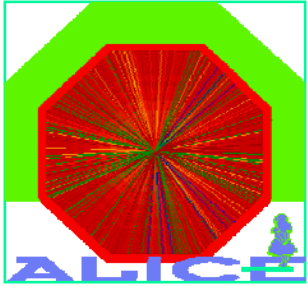


- Monitoring elements (usually histograms) are laid on tabbed windows.
- Ability to monitor from several sources:
 - Online on LDC/GDC
 - Offline (files):
 - ALICE standard data format (ROOT) files on local disk, CASTOR or an http server (GDC events)
 - DATE raw format files on local disk or CASTOR (LDC/GDC)
- Multithreaded execution, GUI fully usable while monitoring
- Single Document Interface, MDI under consideration
- Signal-slot mechanism for handling GUI events
- Can be used as a user-friendly hex event dumper
- Ability to monitor a pre-specified number of events and update the screen automatically after a defined number of events or manually.
- Link to AliRoot code



MOOD for T0 detector (Test Beam)



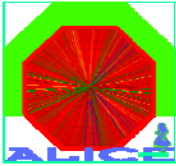


Automatic Data Quality Monitoring

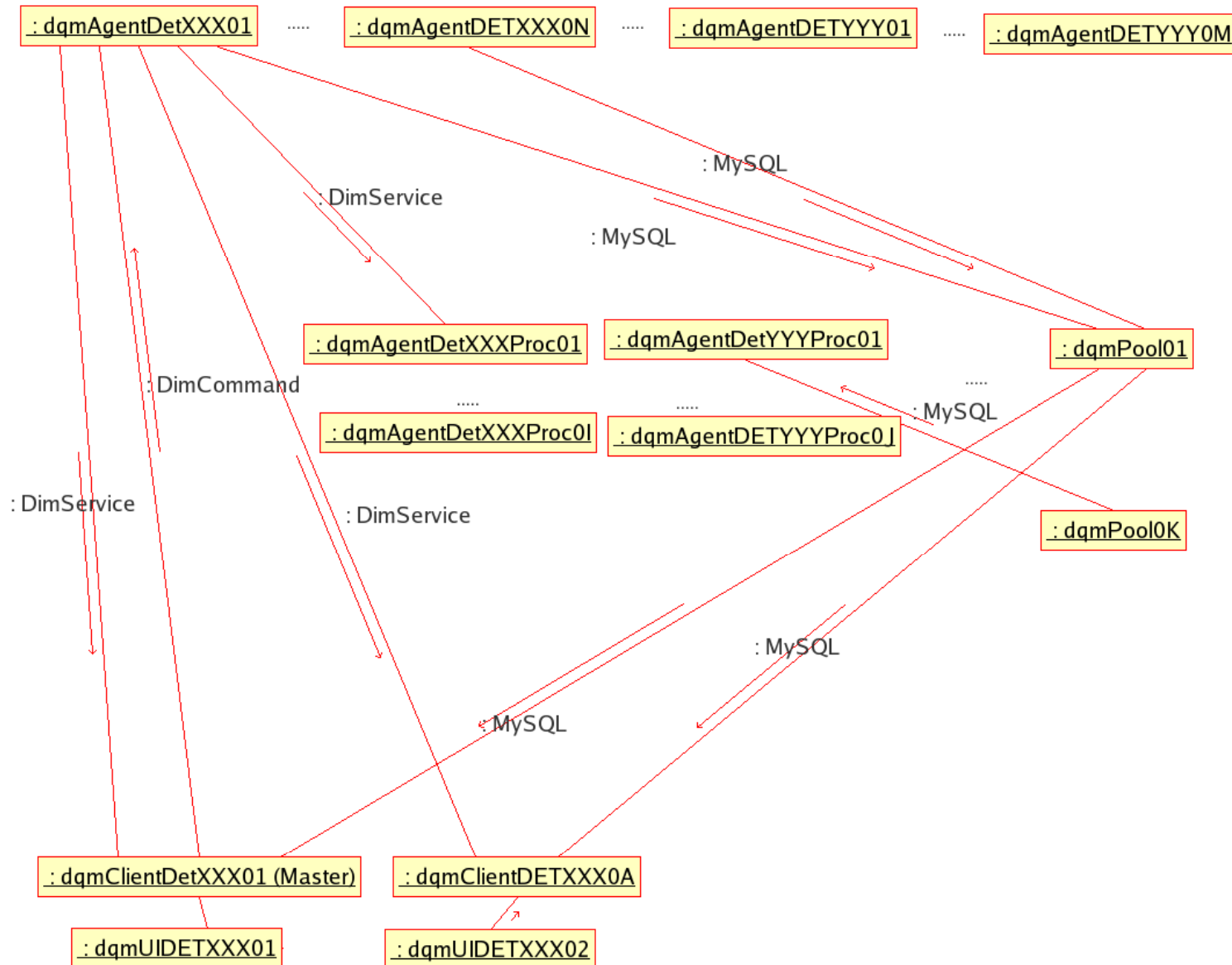
After a lots of DATEs and AFFAIRs, ALICE now has a new...

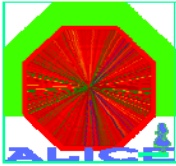
AMORE

Automatic **MO**nito**RI**ng **E**nvironment

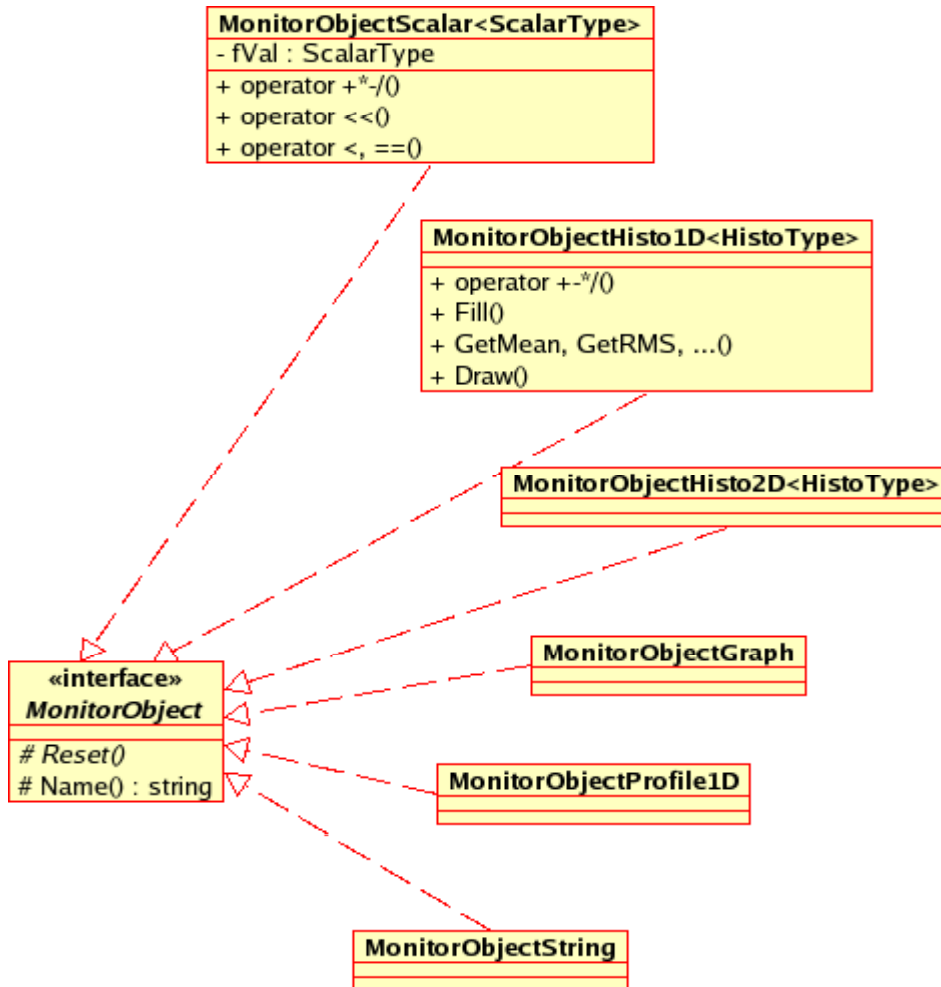


The big picture

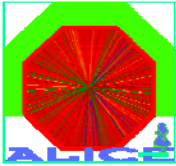




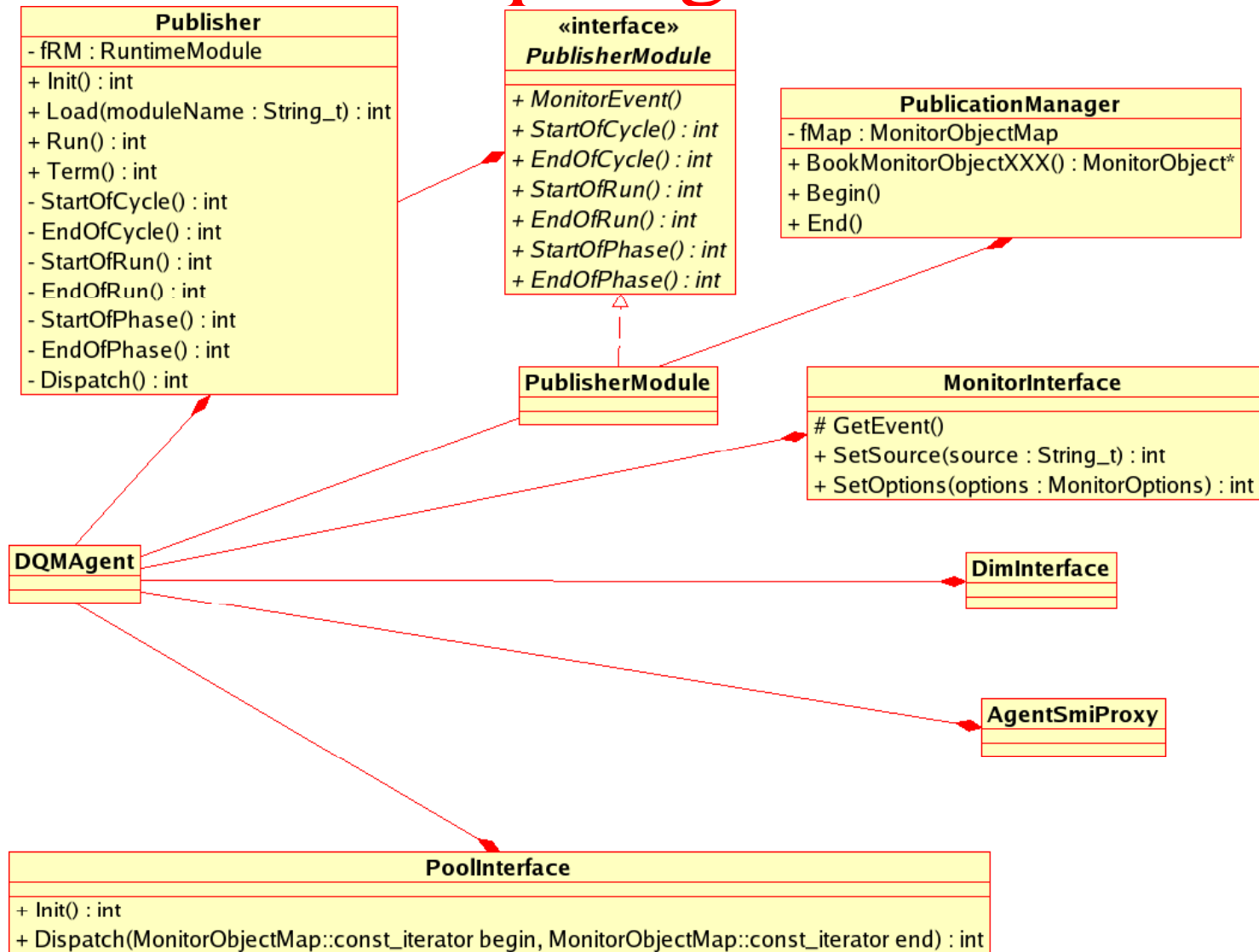
MonitorObjects

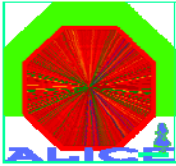


- Template based design, static safe, you cannot e.g. `Fill(x, y, z)` a 1-D histogram. Under testing at the moment.
- All POD types and relevant ROOT types are supported.
- Attributes control the reset/update behavior, done automatically by the framework.
- Short named typedefs exist 😊
- Factories are responsible for the creation/registration/access/destruction of MonitorObjects

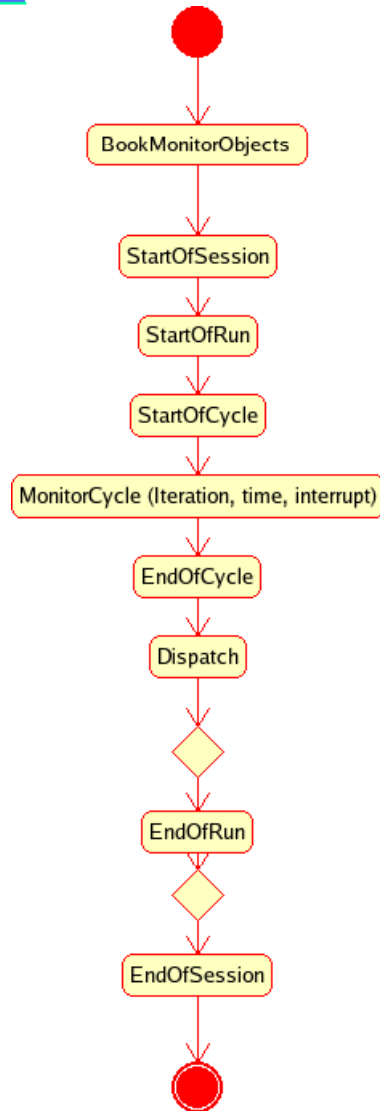


dqmAgent

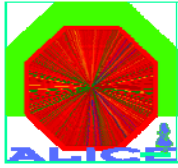




PublisherModule



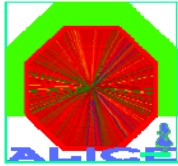
- We follow the “Template” OO Design Pattern. User code overriding this ABC resides in a dynamic library loaded at runtime.
- An agent can exchange the whole module with another one at runtime with time resolution of a MonitorCycle.
- Arbitrary number of modules can coexist in the detector library, loadable by name automatically by ROOT reflexion ☺
- Totally GEEK feature: Shifter can upload “MyCoolModule.C” on the monitoring server where the desired agent runs, compile locally and load! (Stability reasons may eventually not allow this)



Master Client-Agent communication



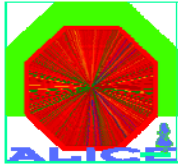
- DIM (Distributed Information Management) system used for interprocess communication, following the ALICE DAQ policy.
- Simple idea: Construct the desired member function call as a string at the source and use Reflexion at the destination to execute it. e.g
Source: `pair<string s, string t>=<“myHisto”, “Rebin(100)”>`
Destination: `gROOT->FindObject(s)->Execute(t)`
- In this implementation the problem is not to give power to the users to take actions on the MonitorObjects on the dqmAgent side but how we can reduce this power!
- Probably filtering of the messages should take place at the client side to disallow doing nasty things on the dqmAgent...
- Messages will be `std::queued` and executed with a time resolution of a MonitorCycle to avoid performance penalties.



Some figures...



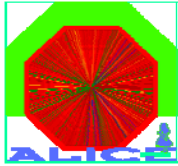
- $\sim 10^9$ MonitorObjects (histograms) have been transferred in several sessions. No memory leaks in the framework core.
- On a P4 @ 2.8 GHz ~ 1.5 million histogram fill operations/sec
- ~ 50 msec to serialize/transfer/deserialize 1000 50-bin TH1Ds
- These numbers are pure inner loop measurements without data: The absolute maximum!
- Limited computing resources: Define and implement the minimum physics requirements for day-1 Run. (event size/channel occupancy, scaler readout, very simple cluster statistics).
- Taking into consideration the time needed for getting an event from DATE monitoring library, decoding and analyzing, rate could drop dramatically to few Hz.
- Client refresh period should be expected to be 1-30 sec in normal conditions.
- So, provision must be taken that every histogram bin has a useful meaning. We may artificially reduce the total number of bins per second to ~ 50 Kbins to avoid network problems!



Visualization



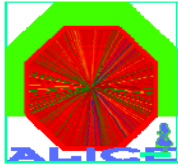
- ROOT GUI heavily used as in the case of MOOD.
- Unlike any other existing DQM framework, in AMORE the C++ code IS the visual layout configuration (no need for elaborate XML/txt configuration) through the magic of Reflexion.
- **Star-Trek feature:** The shifter loads “MyCoolLayout.C” from a USB-stick and voila – a fully customized layout. Not even need to compile in this case!
- **Star-Wars feature** (*with the kind help of ROOT Force*):
 1. Right Click on an empty pad
 2. Put “MyMonitorObject” HERE.(Admittedly still in brainstorming phase, may not lead to anything practical)
- Of course we will provide all the “boring” stuff that every other LHC experiment has, namely predefined sets of visual layouts and a MonitorObject “browser” in case sci-fi fails to deliver in time ☺
- Currently under designing/testing, anticipating the interesting developments in ROOT GUI(builder).



Deployment strategy



- AMORE is going to be distributed as binary RPM package containing shared libraries and executables. (Currently a fully autotools compliant package).
- Detectors will provide their code as class libraries that will be eventually packed in binary RPMs for deployment at Pt2.
- No central source repository for DET code as in MOOD case.
- Single version of ROOT/AliROOT on monitoring servers. If not adequate, DET libraries should be statically linked.



Conclusions



- ✓ MOOD has a new solid implementation.
 - ✓ AMORE is behaving well under test conditions, in a short time we essentially managed to deliver or to make a proof of concept for most of the core functionality of existing DQM frameworks.
 - ✓ A novel approach is used in many aspects.
 - ✓ Makes heavy usage of core ROOT I/O, RTTI, Histogramming, MySQL and GUI.
-
- If something fails to work in practice as expected we will revert back to a well tested solution in any case.
 - As usual, users are the ultimate judges.
 - Stay tuned for the first public release this summer.