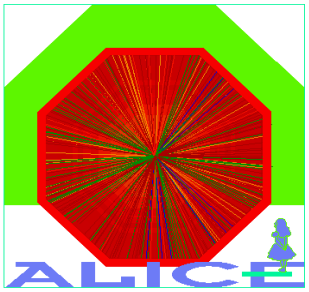


The ALICE-LHC Online Data Quality Monitoring Framework

Filimon Roukoutakis

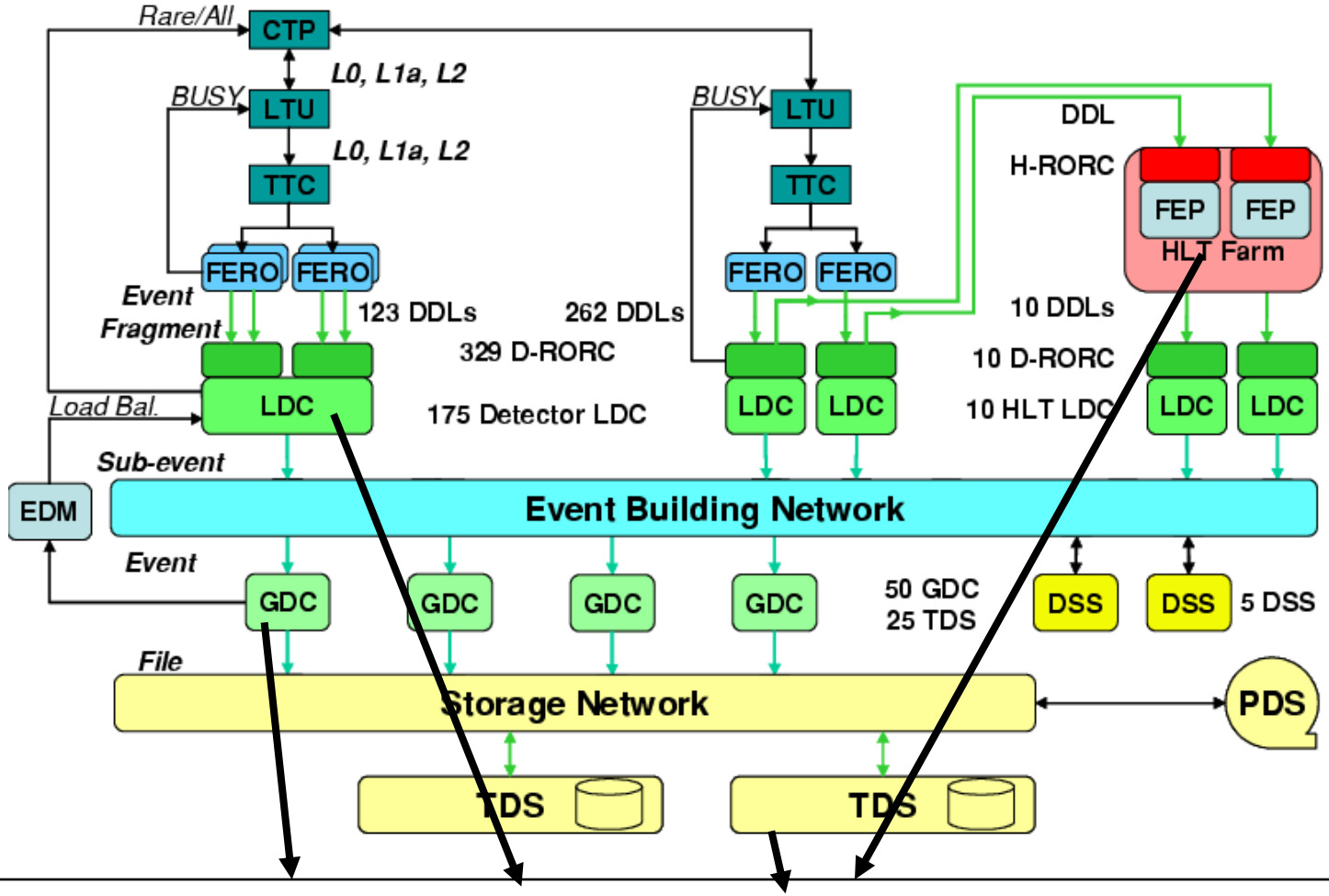
CERN Physics Department

CHEP'07 VICTORIA, BC

International Conference on Computing in High Energy and Nuclear Physics

2-7 Sept 2007 Victoria BC Canada

ALICE DAQ architecture

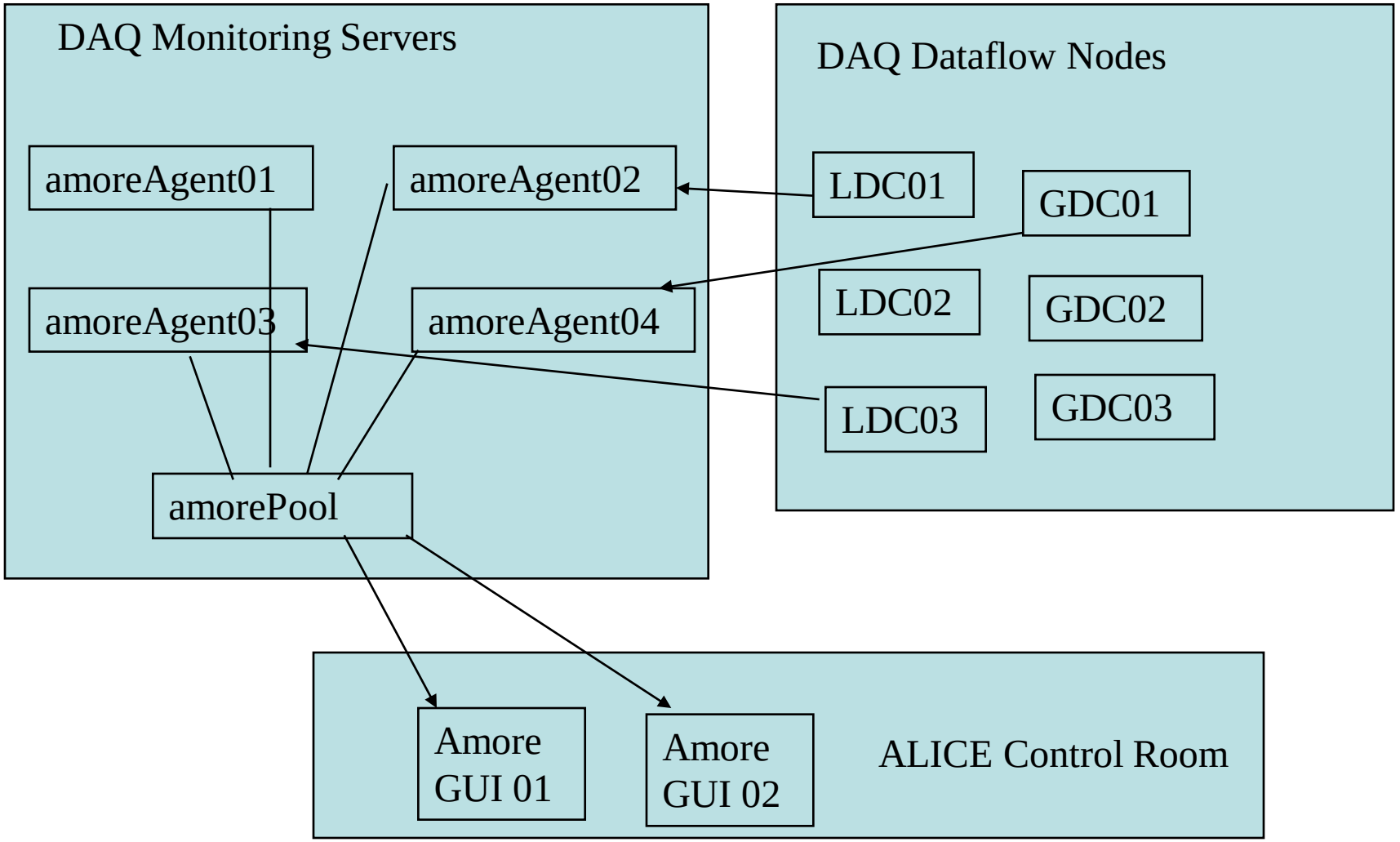


Fundamental design requirements

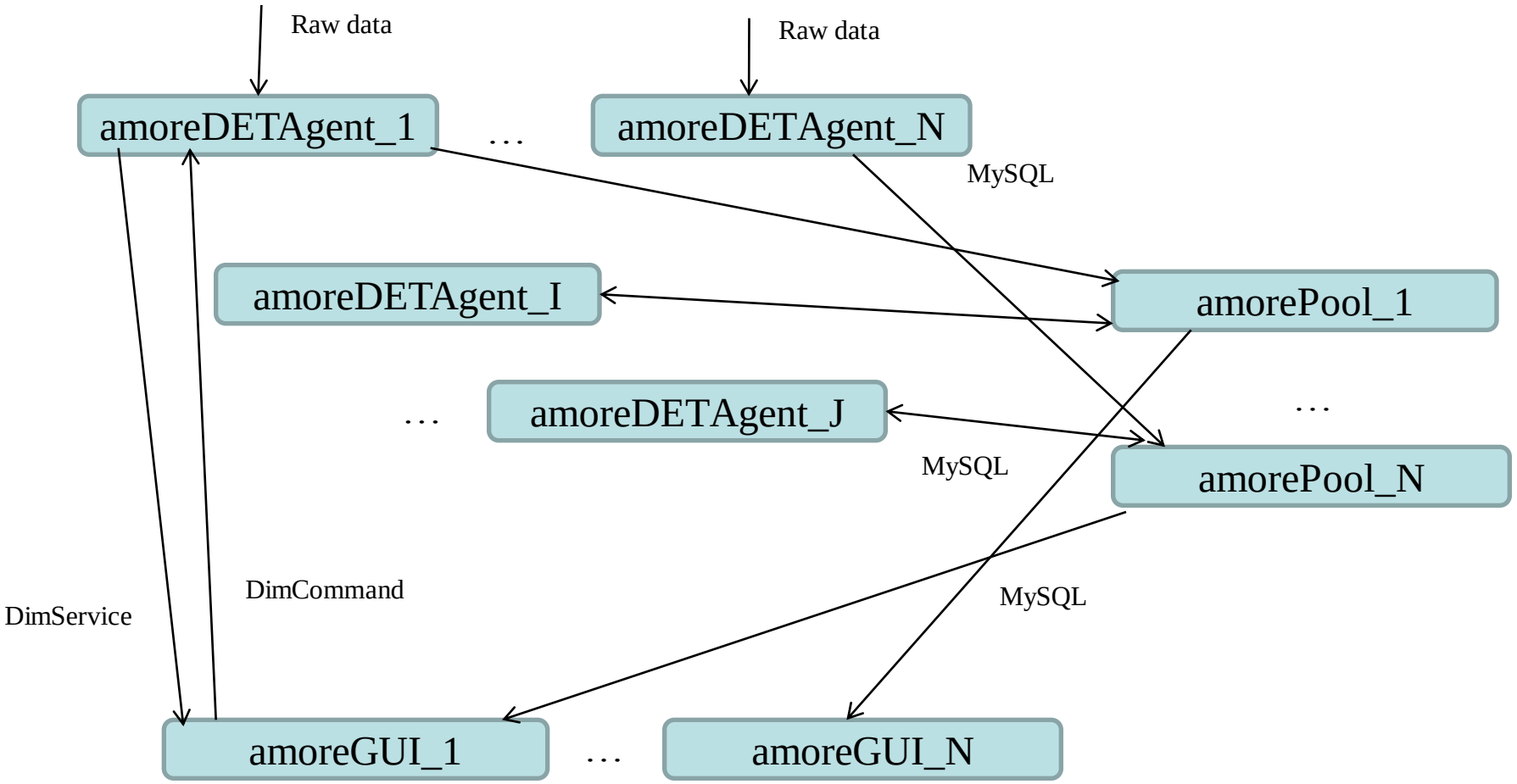
- Follow the publish-subscribe paradigm to allow scalable many-to-many connection between monitoring information producers and consumers. Introduce intermediate pools of monitoring data for full decoupling, thus use a classic 3-tier design
- No dependency on user code, which will probably be stored in the offline CVS. This is done through usage of C++/ROOT reflection

  abc_ptr* myptr=gROOT->GetClass("derived_class_name")->New()
- Use the same IPC/control systems as our DAQ framework, namely DIM and SMI++
- Use MySQL for data pools and configuration
- It should have a "lovely" name, like all the DAQ software (DATE, MOOD, AFFAIR). We called it AMORE (=Automatic MOnitoRing Environment)

The big picture (1)



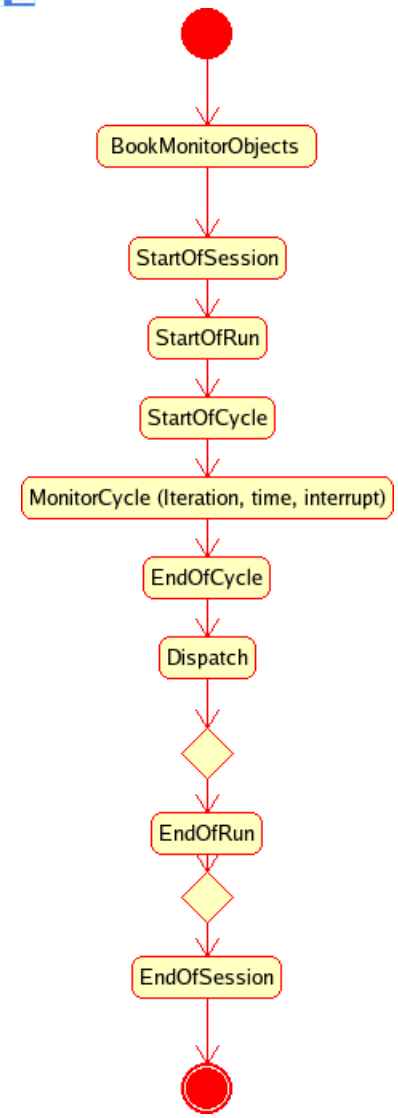
The big picture (2)



Publishers and subscribers

- amoreAgent is an FSM running a single publisher and (optionally) many subscribers
- amoreGUI runs many subscribers
- Each subscriber is associated to a single publisher
- Publisher and subscribers execute user code in classes derived from special ABCs that define the interface to be called by the FSM
- Arbitrary number of publisher and subscriber “modules” with user code can exist in user-created libraries and can be loaded at runtime by the framework

PublisherModule

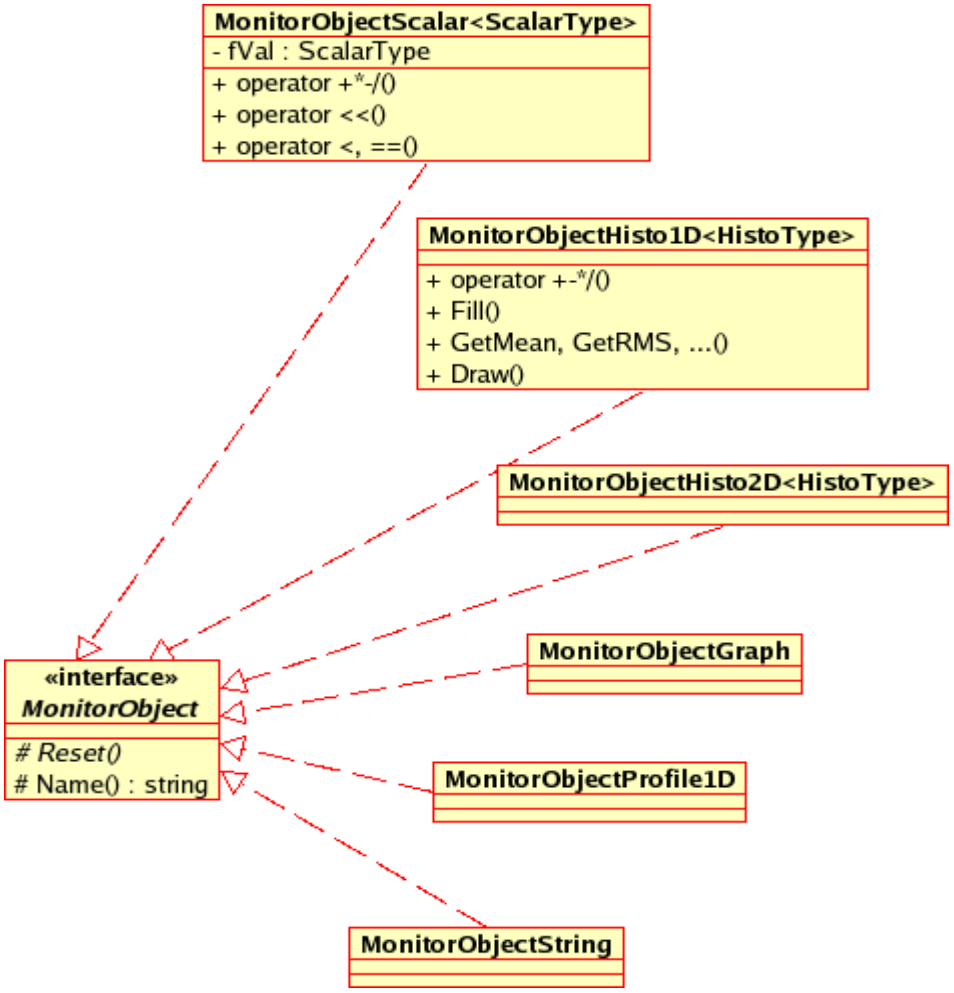


- We follow the "Template" OO Design Pattern. User code overriding this ABC resides in a dynamic library loaded at runtime
- Arbitrary number of modules can coexist in the detector library, loadable by name automagically by ROOT reflection ☺
- An agent can exchange the whole module with another one at runtime with time resolution of a MonitorCycle. (Infrastructure exists, API on client side missing to do it)

RDBMS as amorePool

- Pool implementation is the most critical one for several reasons
- Most problems are common to the ones RDBMS try to solve -> Try to use an expert solution to do the job -> Use MySQL
- Each amoreAgent has its own table in a database to store the published data. When a subscriber "subscribes' to this agent, essentially accesses this table to retrieve monitoring data
- MySQL is also used for the configuration of the system and agent bootstrapping
- The pool implementation is isolated behind an abstract "dispatching" interface. A custom created pool can almost transparently replace the MySQL implementation.

MonitorObjects



- Template based design, static safe, you cannot e.g. Fill(x, y, z) a 1-D histogram.
- All fundamental C++ types and relevant ROOT types are supported.
- Attributes control the reset/update behavior, done automatically by the framework.
- Factories are responsible for the creation/registration/access/destruction of MonitorObjects
- MonitorObjectArray (of fundamental C++ types) under evaluation
- MonitorObjectTree under evaluation in view of a generic approach of the offline for ALICE DQM, based on TTrees.
- MonitorObjectTObj also supported for transport of arbitrary objects to/from amorePools

Data Quality Assesment

- The idea is to reuse MonitorObjects as dq assesment “results” combined with references to other MonitorObjects if needed
- Data Quality checks are friends or members of the MonitorObject type they make sense to be applied on. Examples are: Value within range, chi^2, Kolmogorov,…
- Based on the “value/status” of such a MonitorObject, a process that subscribes to it can take required action, like printing messages, changing screen colors, notifying the ECS,…
- Comparison to reference MonitorObjects under design. We will use centrally stored ROOT files with MonitorObjects for cosmics if there is detector interest.

Deployment

$AMORE -\ (=/opt/amore)

bin -\

The AMORE batch and interactive executables and the setup/configuration shell scripts

include -\

amore -\

*.h (The AMORE API)

lib -\

The AMORE libraries, static and dynamic versions)

- Distributed as either a GNU autotools package to be built by `configure –prefix=/opt/amore; make; sudo make install;` or a binary RPM package for SLC4
- Requirements: ROOT >= 5.16, DATE 6.x (DAQ software framework), MySQL (DATE compatible version)

$AMORE_SITE -\ (=/opt/amoreSite)

lib -\

libAmoreDETCommon

libAmoreDETPublisher

libAmoreDETSubsciber

libAmoreDETUI

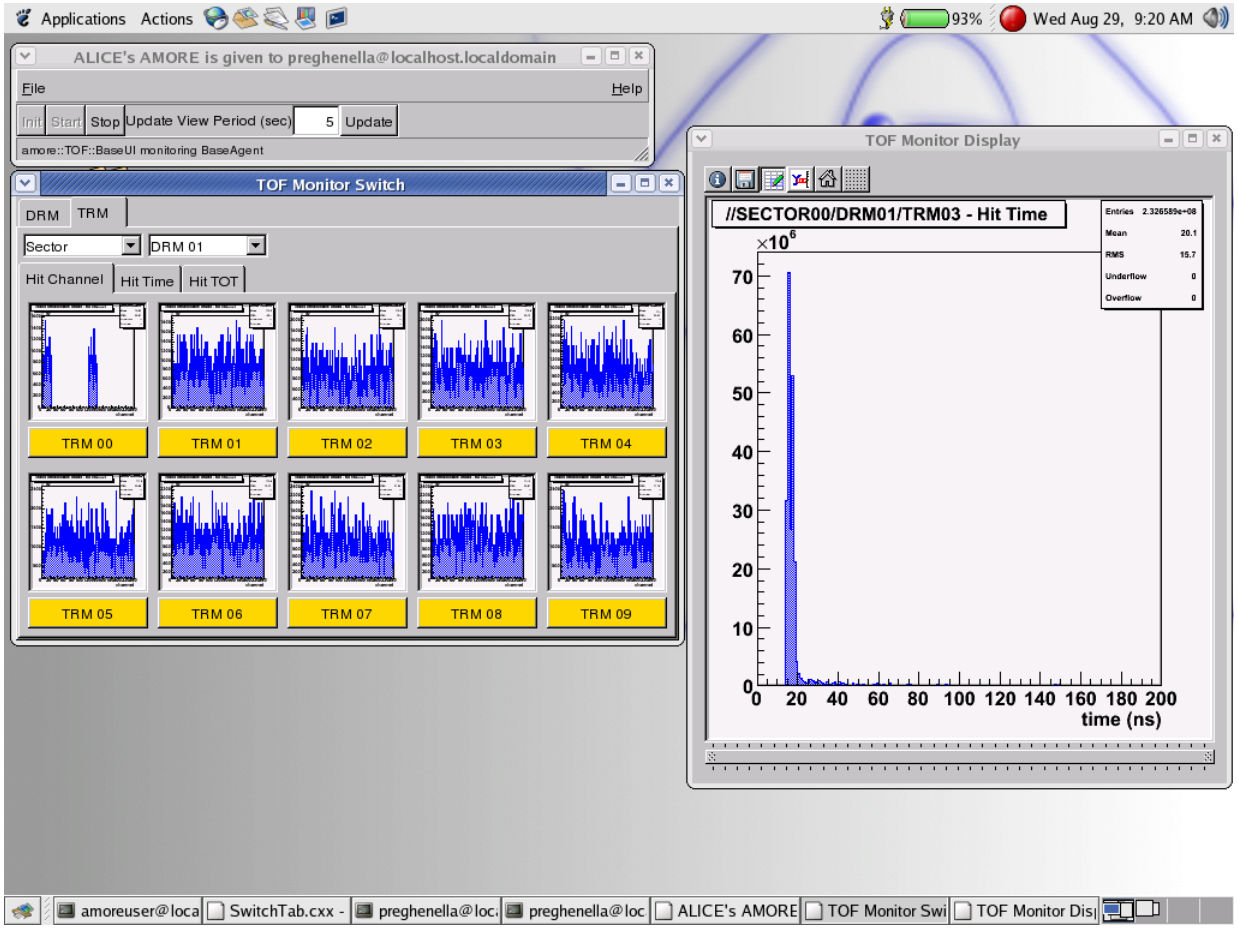
- DET=3 alphanumeric digit official DET code
- A template DET tarball contains example code and makefiles to create the libraries above and package them in RPM These libraries readily link against ROOT/AliROOT by default (SINGLE version existing on DAQ network). They use the AMORE API in $AMORE/include/amore
- No need for a central repository for DET code in DAQ CVS (The offline may however pursue this if desirable).
- All common stuff that is to be used by both servers and clients should be in Common lib. Do not assume that all the libraries exist in every system. All code lives under amore::DET::xxx namespaces, xxx=publisher, subscriber, ui, common

Visualization

- gROOT->LetThereBeLight()
- Arbitrary number of prebuilt visual layouts, loaded by name any time during a run on AMORE GUI.
- 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 Reflection. We can still load configuration values from an external file/db if this is needed by detectors.
- A “browser” of the available MonitorObjects should be eventually in place

AMORE GUI example

A real one…



- Picture courtesy of Roberto Preghenella, ALICE TOF group

Proposal…

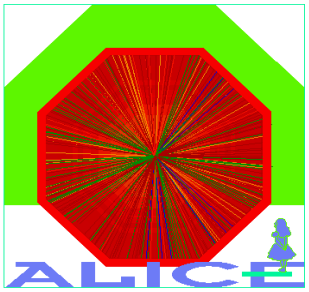
- … for a “DQM4LHC” 1 or 2-day workshop at CERN between December and February
- Present “official” experimental frameworks (if there exist such a thing), detector implementations based on these frameworks and standalone detector “expert” tools created the last years for test beams
- Discuss implementations, user requirements, configuration, visualization, performance, problems (and solutions!)
- Contact Filimon Roukoutakis using std::CERN address to declare interest…

Conclusions

- AMORE is behaving well under test conditions. An amoreAgent can handle O(10^4) MonitorObjects and the MonitorObject handling is already well defined. The GUI seems adequate and easily expandable.

- Several things need to be in place for cosmic run but the needs are better defined now. Further detector requirements will be honored after cosmics but discussion can be ongoing

- Released mid-July, one implementation already existing from TOF detector group



BACKUP

Master Client-Agent communication (to be implemented for cosmics)

- 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 C++/ROOT Reflection at the destination to execute it. e.g

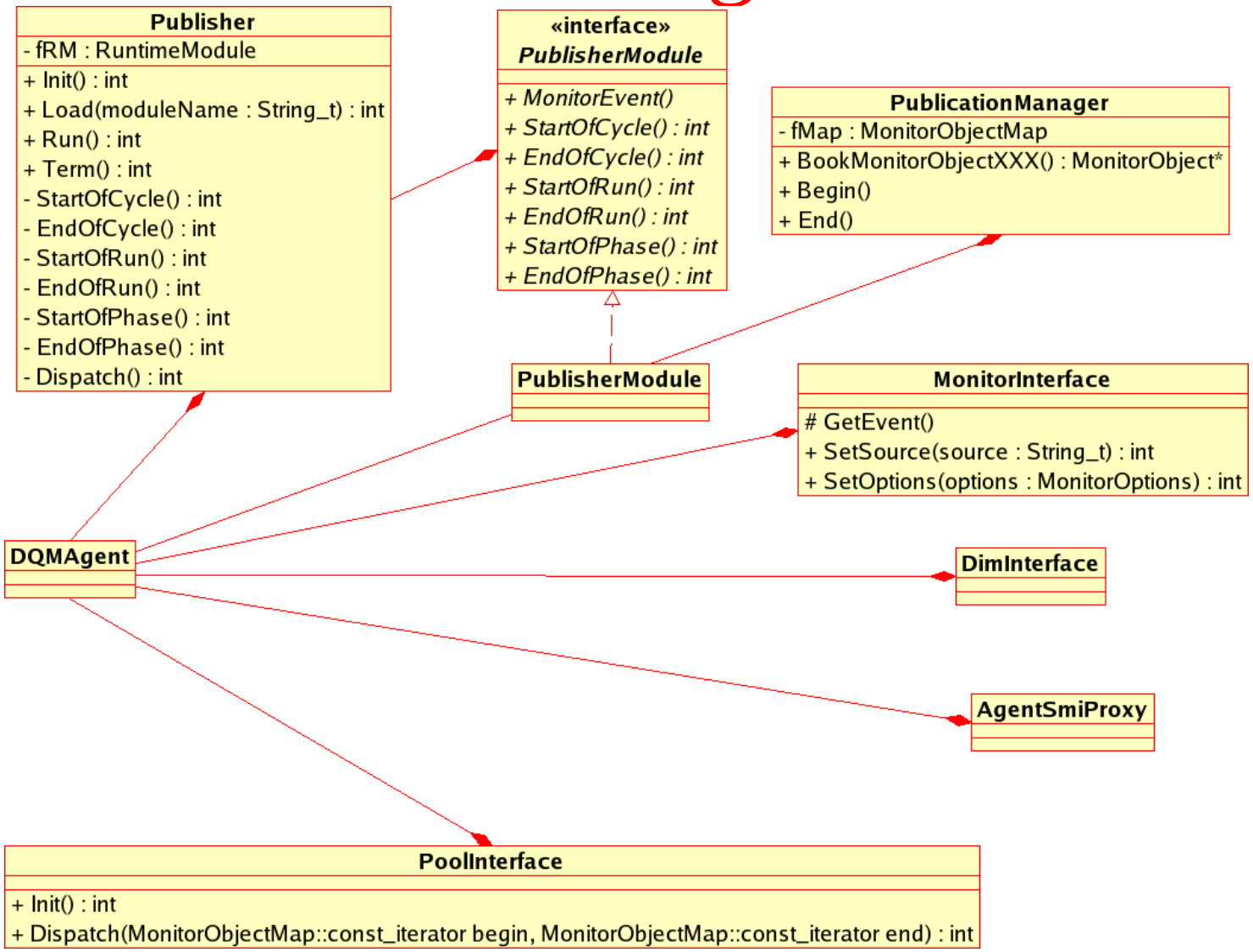
  Source: pair<string s, string t>=<“myHisto”, “Rebin(100)”>

  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 amoreAgent…
- Messages will be std::queued and executed with a time resolution of a MonitorCycle to avoid performance penalties.

Some figures…

- >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, on a current DAQ monitoring server ~5.2 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 5-60 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!

amoreAgent


Publisher
- fRM : RuntimeModule
+ Init() : int
+ Load(moduleName : String_t) : int
+ Run() : int
+ Term() : int
- StartOfCycle() : int
- EndOfCycle() : int
- StartOfRun() : int
- EndOfRun() : int
- StartOfPhase() : int
- EndOfPhase() : int
- Dispatch() : int
«interface»
PublisherModule
+ MonitorEvent()
+ StartOfCycle() : int
+ EndOfCycle() : int
+ StartOfRun() : int
+ EndOfRun() : int
+ StartOfPhase() : int
+ EndOfPhase() : int
PublicationManager
- fMap : MonitorObjectMap
+ BookMonitorObjectXXX() : MonitorObject*
+ Begin()
+ End()
PublisherModule
MonitorInterface
GetEvent()
+ SetSource(source : String_t) : int
+ SetOptions(options : MonitorOptions) : int
DQMAgent
DimInterface
AgentSmiProxy
PoolInterface
+ Init() : int
+ Dispatch(MonitorObjectMap::const_iterator begin, MonitorObjectMap::const_iterator end) : int

Deployment (2)

```
mysql> describe amoreconfig;
+--------------+----------+------+-----+---------+-------+
| Field        | Type     | Null | Key | Default | Extra |
+--------------+----------+------+-----+---------+-------+
| host         | char(32) | NO   | PRI | NULL    |       |
| agentname    | char(32) | NO   | PRI | NULL    |       |
| detector     | char(3)  | YES  |     | NULL    |       |
| dimnode      | char(32) | YES  |     | NULL    |       |
| poolnode     | char(32) | YES  |     | NULL    |       |
| source       | char(32) | YES  |     | NULL    |       |
| defaultmodule | char(32) | YES  |     | NULL    |       |
| configfile   | longblob | YES  |     | NULL    |       |
+--------------+----------+------+-----+---------+-------+
8 rows in set (0.00 sec)

mysql> describe MyAgent1;
+------------+-----------+------+-----+-------------------+-------+
| Field      | Type      | Null | Key | Default           | Extra |
+------------+-----------+------+-----+-------------------+-------+
| moname     | char(64)  | NO   | PRI | NULL              |       |
| updatetime | timestamp | YES  |     | CURRENT_TIMESTAMP |       |
| data       | longblob  | YES  |     | NULL              |       |
+------------+-----------+------+-----+-------------------+-------+
3 rows in set (0.00 sec)
```

Deployment (3)

- An amoreAgent is uniquely defined by name in the DAQ (DIM/SMI) network, i.e. `amoreAgent MyAgent1` starts a uniquely defined actor with well defined initial configuration by the db

- Several amoreAgents can run the same module. The idea is to produce the same set of MonitorObjects from different DAQ sources concurrently. Therefore, a MonitorObject in the amorePool is uniquely defined by the pair <agentname (=table name), moname (=row)>

```
mysql> select * from amoreconfig;
+---------+-----------+----------+---------+----------+--------+---------------+------------+
| host    | agentname | detector | dimnode | poolnode | source | defaultmodule | configfile |
+---------+-----------+----------+---------+----------+--------+---------------+------------+
| pcald41 | MyAgent1  | DET      | pcald30 | pcald41  | :      | MyModule1     | default    |
| pcald41 | MyAgent2  | DET      | pcald30 | pcald41  | :      | MyModule1     | default    |
| pcald41 | MyAgent3  | DET      | pcald30 | pcald41  | :      | MyModule2     | default    |
| pcald41 | MyAgent4  | DET      | pcald30 | pcald41  | :      | MyModule2     | default    |
+---------+-----------+----------+---------+----------+--------+---------------+------------+
4 rows in set (0.00 sec)
```

UIModule

- Similar to PublisherModule, runs on client side, namely amore GUI.
- The important thingy here is the “Update” member function (acting like a callback) which on specified interval, a button press or a notification from agents (not existing yet) fetches an updated list of MonitorObjects from the pool, postprocess and displays on screen.
- ConstructGUI contains user code to build the ROOT GUI.
- On Update, TCanvas::cd(), MonitorObject::Draw()
- Subject to change as soon as a GUI API is well defined. TRootEmbeddedCanvas will be replaced with AmoreCanvas with enhanced custom functionality in the form of context menus.