CMS Reconstruction and Analysis: an Object Oriented Approach

CMS Collaboration

presented by

Vincenzo Innocente

CERN, Geneve, Suisse

Object Oriented technologies have been chosen to design and implement the first prototype of the CMS Reconstruction and Analysis software. The architecture is based on an application framework in which physics software modules can be plugged in a flexible and customisable way. A key feature of this framework is a mechanism for reconstruction on demand which has been already subjected to intensive prototype work.

Key words: HEP; LHC; CMS; Object Oriented; C++; ODBMS; Event Reconstruction;

1 Introduction

The CMS experiment [1] is one of the two LHC general purpose experiments. Data taking is scheduled to start in 2005 and will last at least ten years. CMS software requirements and computing resources [2] will far exceed those of any existing high energy physics experiment, not only for the complexity of the detector and of the physics task but also for the size of the collaboration and the long time scale. Therefore, software should be developed keeping in mind not only performance but also modularity, flexibility, maintainability, quality assurance and documentation. Object Orientation has been identified as the enabling technology, since it directly addresses this kind of problems.

The overall design of the CMS Software Architecture is motivated by the following underlying principles:

– Multiple Environments: Various software modules must be able to run in a variety of environments as different computing tasks are performed (examples of

these environments are level 3 triggering, production reconstruction, program development, and individual analysis);
- Migration between environments: a particular software module may start out being developed for one environment, then later used in other unforeseen environments as well;
- Dispersed code development: the software will be developed by organisationally and geographically dispersed groups of part-time non-professional programmers – only some portions of the code will be written by computing professionals;
- Flexibility: not all software requirements will be fully known in advance, therefore the software systems must be adaptable without requiring total rewrites;
- Ease of use: the software systems must be easily usable by collaboration physicists who are not computing experts and cannot devote large amounts of time to learning computing techniques.

These requirements on the software architecture result in the following overall structure for the CMS software:

- a customisable application framework for each of the computing environments;
- physics software modules with clearly defined interfaces that can be plugged into the framework;
- a service and utility Toolkit that can be used by any of the physics modules.

The framework defines the top level abstractions, their behaviour and collaboration patterns. It will comprise two components: a set of classes that capture CMS specific concepts like detector components and event features and a control policy that orchestrates the instances of those classes taking care of the flow of control, module scheduling, input/output, etc. This control policy will be tailored to the task in hand and to the computing environment.

The physics and utility modules will be written by detector group and physicists. The modules can be plugged in the application framework at run time, independently of the computing environment. One can easily choose between different versions of various modules. The physics modules will not communicate with each other directly but only through the data access protocols that are part of the framework itself.

The service and utility toolkit will consist of two major categories of services: physics type services (histogrammers, fitters, physics calculation routines) and computer services (data access, inter module communication, user interface, etc.) and will most probably not be CMS specific.

2 Analysis Model

CMS physicists will perform their analysis activities (event selection, physics analysis, detector studies, calibrations) mainly using "high-level" concepts (particles, tracks, jets, etc.), even though some studies will require direct access to "low-level" information (raw or preprocessed data coming from the DAQ or the simulation program).

We define reconstruction as the data reduction process which creates, out of low-level information, the objects which represent these high-level concepts. In the following we will call these objects "Reconstructed Objects" (RecObj's). At this stage the concept of Reconstructed Object encompasses all possible objects used in analysis, not only the ones provided by the CMS reconstruction code but also the user defined ones. Therefore "RecObj" may not identify a unique object (super-)type.

The lowest level RecObj's are built directly from the Raw-Objects (objects directly provided by the DAQ). The highest level RecObj is a concise description of the event itself (Event Tag).

A major activity in Analysis is the definition of the event sample to be used. We call Event Filter (or Event Selection) the process which decides if an event belongs to a certain sample according to properties of the event itself defined using RecObj's belonging to the event.

3 Analysis Life-Cycle

Data coming from the detectors and corresponding to a particular beam crossing triggered by Level-1 (Event) are organised in an on-line event filter unit (Level-2 and -3 triggers) and permanently stored if they satisfy Level-2 and -3 trigger conditions. The CMS simulation program will produce data organised in a way similar to those coming from the DAQ.

All events passing Level-3 are classified according to some physics selections and some additional information may be stored permanently. Physics analysis and detector studies (calibration, monitoring, etc.) are performed using these events. These activities usually include further reconstruction and event selection.

Analysis is performed in cycles. In each cycle an event sample is selected and further studied until a more refined Event Filter is defined. Such new Event Filters select the event sample to be used in the next cycle. A particular analysis cycle is not only limited to a specific event sample but often uses only certain attributes of a reduced set of RecObj's.

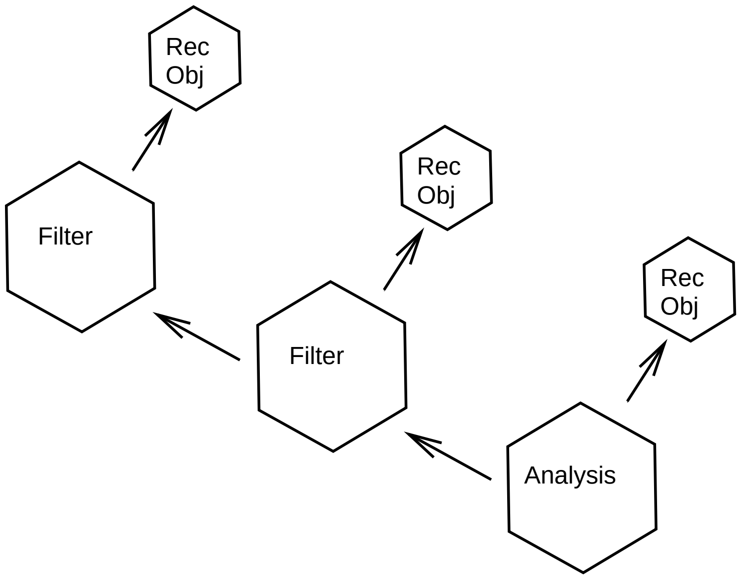

Fig. 1. Analysis life-cycle represented as a sequence of Event Filters (arrows indicate the direction of requests, not data flow)

Figure 1 shows how the Analysis life-cycle can be seen as a sequence of Event Filters each using the result of a previous Event Filter and the information of some Reconstructed Objects. In particular Level-2 and Level-3 triggers can be seen as the very first stages of this chain of Event Filter units.

¿From such a model we can thus identify two major classes of objects involved in CMS data processing and analysis: Reconstruction units which create reconstructed objects, and Event Filters which use these reconstructed objects to classify and select events.

4 Reconstruction Model

The reconstruction process is a collection of independent “Reconstruction Units” each reconstructing a given RecObj (sub-)type. The only dependencies among these units are that RecObj’s can be built using other lower-level RecObj’s. Reconstruction Units can share common sources of information or services from common utility tools (which are re-usable components of CMS reconstruction software). Sources of information for a Reconstruction Unit are:

– DAQ or simulation program;
– Other Reconstruction Unit (only through the created RecObj’s);
– Detector Description;
– Detector Status (including calibrations);
– Reconstruction algorithm parameters;
– User Options.

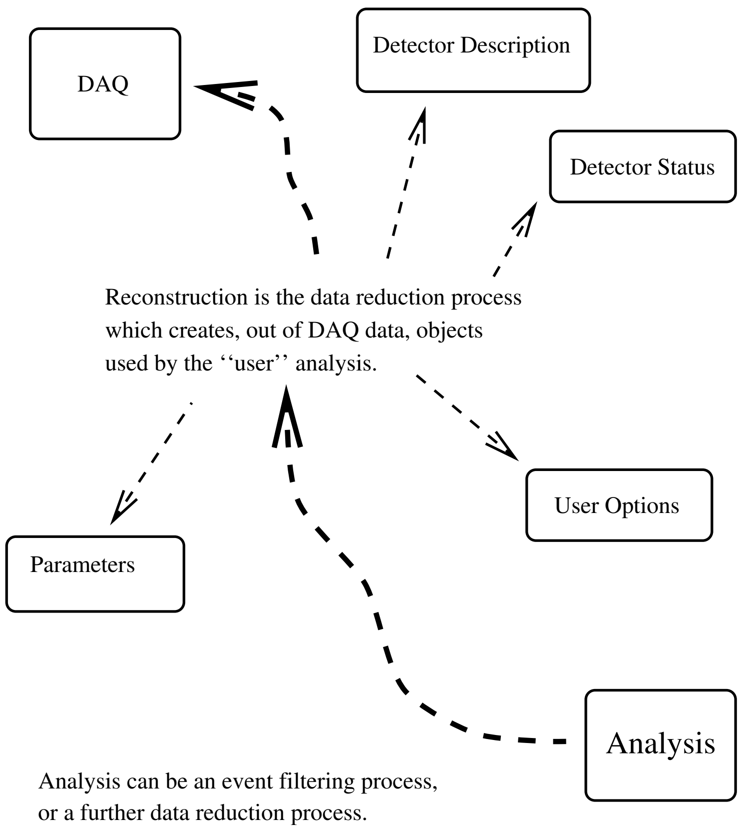

Fig. 2. Sources of information for a Reconstruction Unit (arrows indicate the direction of requests, not data flow)

This is schematically shown in Figure 2. All this information is critical for correct reconstruction and therefore should undergo strict version control. It should always be possible to relate reconstructed objects to the information used to reconstruct them. An ODBMS is ideal to manage such information and such relations. Storing them into the same federation will allow direct associations among them instead of using relational techniques.

A Reconstruction Unit usually performs the following activities:

– Pattern recognition: the process that, starting from collections of lower level RecObj's, constructs the RecObj's of interest. Usually it makes use of time consuming combinatorial algorithms.
– Fit: the process which computes the physical or geometrical attributes of a RecObj starting from information provided by its constituent RecObj's.
– Correction: the process whereby additional detector information (calibrations, alignments, etc), not available (or meaningful) at constituent level, is used to build or fit a RecObj.

Although these three basic activities can be interleaved and mixed depending on the actual reconstruction algorithm, it is current practice to split them in two parts:

- Object creation: it is the process which creates "stable" RecObj's in terms of, at least, their constituents. It includes pattern recognition and "zero-order" corrections. In general changes to algorithms or information used in this process do not ensure that the resulting RecObj's will keep their "identity". Full consistency requires that previous RecObj's be deleted if this process needs to be re-run.
- Object updates: it is the process whereby the physical and/or geometrical attributes of a RecObj are updated. It includes fit and "high-order" corrections. It can include some "non-combinatorial" pattern recognition (object extension). In general, algorithms and information used in this process do not modify the identity of a RecObj but only its attributes. If an update process is run a new version of a RecObj is created.

It seems natural to assign the responsibility of object creation to the Reconstruction Unit and the responsibility of object update to the object itself.

5 Analysis and Reconstruction Framework

Having described the analysis and the reconstruction model, we are now in the position to enumerate the components of the CMS Analysis and Reconstruction Framework.

- Persistent storage manager: the analysis model requires explicitly that data are persistently stored. Both event and non-event data have to be managed. It will definitely be an application based on an ODBMS. It will incorporate several distinct mechanisms to efficiently retrieve the required objects in response to a user query.
- Reconstruction Framework: the system which provides the user with the requested RecObj's and steers their reconstruction according to the user request and the availability of already stored and/or still valid objects. It is intimately connected to the persistent storage manager being its major client. It should be highly flexible, enabling the user to define his own RecObj's and Reconstruction Units without affecting the architecture of the framework itself.
- Analysis Framework: manages the chain of Event Filters which constitute an analysis cycle; uses the Reconstruction Framework to obtain the required RecObj's.
- User Interface (not just a GUI): the user should be able to customise his application and/or change any "input" parameter. It can use the persistent storage manager as interface to the Reconstruction and Analysis framework.
- Event Visualisation Framework: manages the graphical representations of the detector and of the various RecObj's. It can be integrated with the Reconstruction Framework to form a fully interactive reconstruction application.
- Statistical analysis tools: mainly histogramming tools, and corresponding graphical packages.

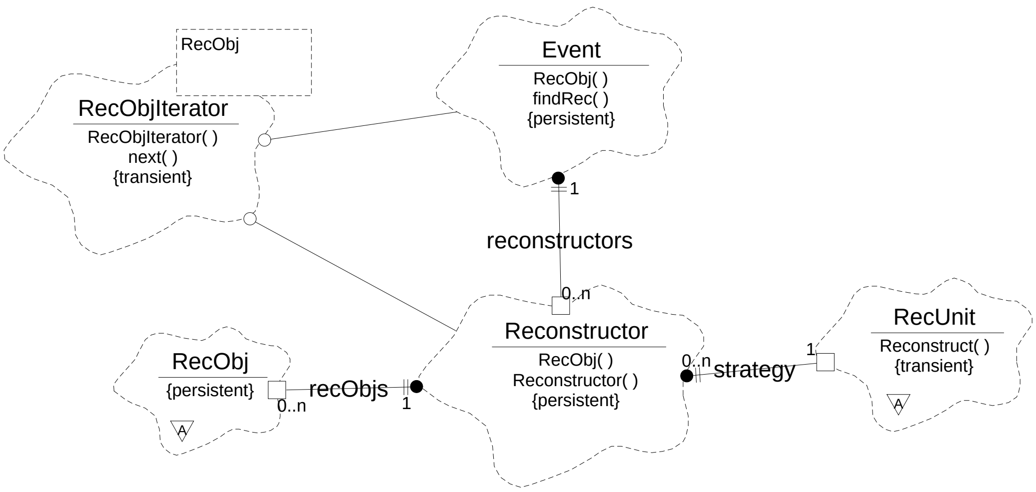

Fig. 3. Class diagram showing the classes collaborating in the reconstruction on demand mechanism

– Fitting tools: such as MINUIT and related routines.
– Presentation tools: whatever is required to represent Physics Results.

6 Reconstruction Prototype

To validate the design of the architecture CMS has initiated prototyping activities for all the major mechanisms involved.

As an example we describe here the “Reconstruction on Demand” mechanism which ensures that the requested RecObj’s (and only those) are properly reconstructed and up-to-date. Such a mechanism is critical to the implementation of the CMS computing model [2] which foresees a dynamic balance between computation and retrieval of information to efficiently use the computing resources (processing power, storage and networks) available at a given moment from a given place. It also matches the requirements of the online event filter farm [3] where a trigger decision has to be taken as soon as possible without pre-emptive handling of data that will eventually not be used.

The reconstruction on demand mechanism has been intensively investigated including its relationships with the persistent storage manager and the online environment.

Figure 3 shows the classes collaborating in this mechanism and their relationships:

– **Event:** The *entry point* to all information about an event.
– **RecObj**: An abstract class which needs to be specialised in the various concrete types (track, electron, jet, etc.).
– **Reconstructor:** For each type of RecObj the event has a reconstructor which

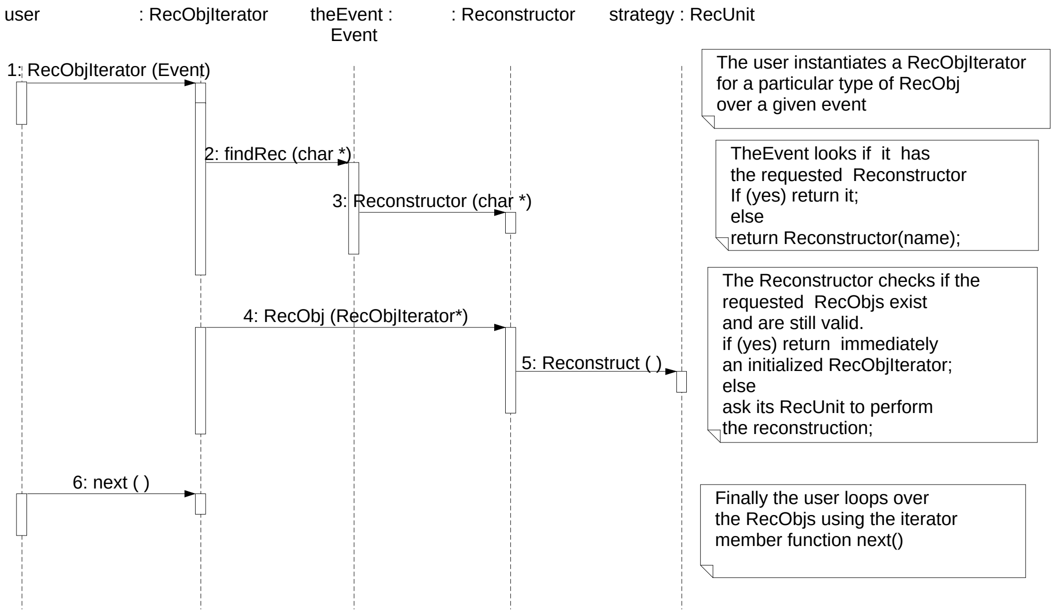

Fig. 4. Scenario diagram representing the reconstruction on demand mechanism

manages the corresponding RecObj's and the various reconstruction activities. The Reconstructor class is concrete and does not need to be specialised for each RecObj concrete type. In the current design it is identified by the name of the corresponding RecObj.

- **RecUnit:** The actual Reconstruction Units will be specialisations (by inheritance) of this class. The relationship Reconstructor–RecUnit implements a well known *Strategy Pattern* [4].
- **RecObjIterator**: The user entry point to this mechanism. It is a parametrised class (a C++ template class) with instances for each concrete RecObj type. It is an extension of the classical iterator with the additional feature that the invocation of its constructor will trigger the "reconstruction on demand" mechanism for its parameter type, as shown in the scenario diagram of Figure 4.

This architecture has several advantages:

- There is no need to modify the Event class (nor any other class in the top level architecture) when a new concrete RecObj subclass is introduced.
- RecObj's are reconstructed when needed and if needed. Their reconstruction is triggered by the object which actually uses them: there is no need for a steering routine where the order in the reconstruction process has to be defined and the dependencies among reconstruction units has to be known in advance.
- Reconstructor can be versioned: the event entry point (the Event object) remains always the same while different versions of reconstructed objects are managed at Reconstructor level.
- In the present prototype RecUnit is a transient class. It can easily become a persistent class with the responsibility of keeping a record of all information actually used by the reconstruction unit itself.

7 Conclusions

The complexity of the computing tasks of the CMS experiment requires its software to be developed using methods that will ensure not only its correctness but also flexibility and ease to use. This will cope with the inevitable changes in requirements and detector configuration, with a large collaboration and with a long time scale.

In the last two years a series of studies and prototypes have shown how Object Oriented technologies are the best candidates to satisfy those requirements.

The design of a Reconstruction and Analysis framework, based on Object Oriented technologies, has been started and the first prototyping work already shows the effectiveness of our choice.

References

[1] "CMS - The Compact Muon Solenoid, Technical Proposal", CERN/LHCC 94-38, LHCC/P1, CERN 1994

[2] "The Compact Muon Solenoid, Computing Technical Proposal", CERN/LHCC 96-45, CERN 1996

[3] A. Kruse, "CMS Online Event Filter Farm Software", Paper presented at this conference.

[4] E. Gamma et al. "Design Patterns", Addison-Wesley, Reading, Massachusetts, 1994.