

Chapter 1

Software Components

Editor(s): Silvestris, Yagil

1.1 Overview

Editor(s): Lucia, Avi

FIXME: one page, the current content need to be changed

Short description of the frameworks. Briefly describe COBRA here and cite previous reports, since COBRA is used extensively in the PTDR studies. Then also describe the new framework and event data model in general terms, with full details in Appendix ??.

1.2 Software Domain Decomposition

Editor(s): Lucia, Avi

FIXME: one page

1.3 Framework

Editor(s): Liz,

FIXME: in total 7 pages

1.3.1 Introduction

The primary goal of the CMS Framework and Edm is to guide developers into producing reconstruction and analysis software that is easy to use. For example developers are encouraged to make their persistent objects as simple as possible in order to completely remove the need for analysis ntuples and the code used to make them. All results should persist in a form directly suitable for analysis purposes. Automation is another way to insure consistency and ease of use. The Framework provides ways to guarantee reproducibility, by automatically maintaining and recording sufficient provenance information for all application results so that the developers don't have to.

1.3.2 Design Principles

The design of the core software was guided by a set of requirements intended to make it easier for physicists to contribute to the trigger and reconstruction programs, to simplify (event) data management, and to make analysis of reconstructed data reproducible.

One guiding requirement was the need allow independent development and verification of distinct elements of triggering, reconstruction, and analysis. The concept of an event-processing *module*, each of which encapsulates a unit of clearly defined event-processing functionality, was introduced to support this goal. Such modules are not allowed to communicate directly with each other, which allows them to be independently tested and reused.

A second guiding requirement was to avoid the need for physicists performing analysis to write their own ntuples. To support this goal, the natural *persistent* form of the event data (see 1.3.5.1) was made to be a Root tree. Restrictions on the form of the products of the trigger and reconstruction programs are enforced so that this may be achieved. This format allows those who want to use Root to perform analysis to use the output of the reconstruction program directly, and decreases the need for physics groups or individuals to create their own ntuple formats, and so decreases the need for special data handling of such ntuples.

The third guiding requirement was to assure that analyses are reproducible; to support this goal, two steps were taken. First, once the product of any step of triggering or reconstruction is put into the *Event* that that object is considered immutable. Second, the event processing application has been made to automatically record the necessary provenance information (see 1.3.6.5) for each element of reconstruction, without the need for physicists writing reconstruction code to take any special action.

1.3.3 Major Components

The EDM framework design is based on layers. At the highest layer is the *EventProcessor*. The *EventProcessor*'s job is to initialize the job and then process the requested number of Events. The *EventProcessor* parses the configuration information provided by the user, creates the unscheduled framework modules which are listed in the configuration and then uses the *ScheduleBuilder* to create the scheduled modules and the internal memory structure that represents the schedule of modules to be executed for each Event. The *EventProcessor* uses the *ScheduleExecutor* to run the sequence of Event based modules and uses the *EventSetupProvider* to manage all the non-Event data services.

1.3.4 Services

To be able to fully process an Event requires additional information outside of the Event itself (*e.g.*, magnetic field measurements). The Non-Event data is data whose 'interval of validity' (*IOV*) is longer than one Event. We have two types of *IOVs* which are distinguished by whether or not the DAQ system initiated the interval of validity transition. *IOVs* initiated by DAQ (such as the Event or a Run transition) are to be handled by the Event system. All other *IOVs* are handled by the *EventSetup* System. The *EventSetup* system provides a unified access model for all services that deliver Non-Event data.

1.3.4.1 *EventSetup* System

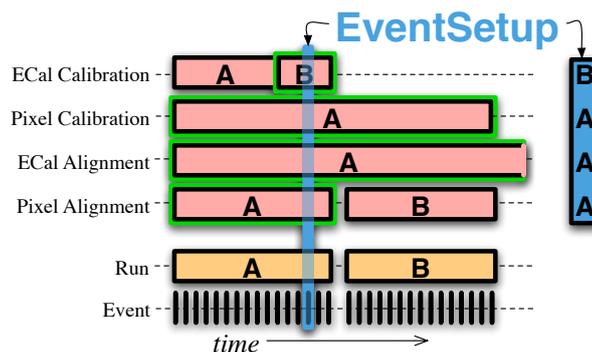


Figure 1.1: The *EventSetup* is formed from the Records that have an *IOV* that overlaps with the moment in time that is being studied.

The *EventSetup* provides a uniform access mechanism to all data/services constrained by an *IOV*. Figure 1.1 shows the main concepts for the *EventSetup*, which are:

1. *Record*: holds data and services which have identical *IOVs*.

2. *EventSetup*: holds all *Records* that have an *IOV* which overlap with the 'time' of the Event being studied.

1.3.4.2 *EventSetup*

The *EventSetup* C++ class provides access to the various *Records* it contains. If the requested *Record* is not available, a C++ exception will be thrown and the Framework will take appropriate action. In this way physicists can safely assume that the *Record* they want exists. In addition to access to *Records*, the *EventSetup* provides access to information about the 'instance in time' for which the *EventSetup* is describing (e.g., the run and event number).

1.3.4.3 *Records*

Records provides safe, read-only access to the objects it contains. This is done in a method analogous to data access from the *Event*. A *Record* also provides access to its interval of validity (*IOV*).

1.3.4.4 Contents of a *Record*

The *EventSetup* system sets no requirements on the C++ class of an object which may be placed in a *Record*. The only restriction is the lifetime of the objects within a *Record* is only guaranteed to be as long as the *IOV* for which the *Record* is appropriate (e.g., if the *IOV* of the data is only good for one run, then the object holding the data is only guaranteed to be available for that one run). This does not mean that an object within a *Record* can not be reused across an *IOV* transition (e.g., from one run to the next), it only means code that reads the object from a *Record* should not assume that it will be reused.

In the case where the C++ type of a data or service is only meant to come from one *Record* (the ECAL pedestals data only come from the ECAL pedestal *Record*), then the 'default' *Record* can be declared at compile time. If a 'default' *Record* has been declared for the C++ class holding some data, then users can access that data directly from the *EventSetup* without having to specify the *Record*. E.g., a physicist can ask for the ECAL pedestal data directly from the *EventSetup* rather than first asking the *EventSetup* for the ECAL pedestal *Record* and then asking that *Record* for the data.

1.3.4.5 *EventSetup* System Components

The *EventSetup* system design uses two categories of components to do its work: *ESSource* and *ESProducer*. These components are configured using the same configuration mechanism as their *Event* counterparts, i.e., via the *ParameterSet* system.

ESSource: An *ESSource* is responsible for determining the *IOV* of a *Record* (or a set of *Records*). The *ESSource* may also deliver data/services. An *ESSource* normally reads its information from a 'persistent store' (e.g., a database) although it is not required to do so. E.g., the ECAL pedestals will be delivered via an *ESSource* that reads the appropriate values from a database.

ESProducer : Conceptually, an *ESProducer* is an algorithm whose inputs to its algorithm are dependent on data with *IOVs*. The Producer's algorithm is run when ever the *IOV* of the *Record* to which it is placing data changes. E.g., an *ESProducer* is used to read the ideal geometry of the tracker as well as the alignment corrections and then create the aligned tracker geometry from those two pieces of information. This *ESProducer* is told by the *EventSetup* system to create a new aligned tracker geometry whenever the alignment changes.

1.3.4.6 Dependent Records

Sometimes an algorithm in the *EventSetup* is dependent on data coming from more than one *Record*. For example, the aligned tracking geometry is dependent on the 'ideal geometry' and on the tracking alignment values. In such a case, the *Record* used by that algorithm needs to be declared 'dependent' on the other *Records*.

The *IOV* of a dependent *Record* is the intersection of the *IOV* of all the *Records* to which it depends. E.g., the *IOV* of the aligned tracking geometry must change when either the *IOV* of the ideal geometry or of the alignment values changes. The *EventSetup* system guarantees that the proper relationships between the *IOVs* is preserved.

Dependent *Records* allow access to only those *Records* to which they are dependent. In this way, the *IOV* dependencies between *Records* can be enforced (if you can not read data from another *Record* then you can not be dependent on that *Record*).

1.3.5 Data Access Model

The data access model is centered around the Event. The Event holds all data that was taken during a 'physics event' as well as all data derived from the taken data. Auxiliary information needed to process an Event is accessed via the *EventSetup*.

Events are processed by passing the *Event* through a sequence of modules. The exact sequence of modules is specified by the user. When an *Event* is passed to a module, that module can get data from the *Event* and put data back into the *Event*. When data is put into the *Event*, the provenance information about the module that created the data will be stored with the data into the *Event*.

1.3.5.1 Event

The *Event* class¹ represents the observed and inferred products of a single interaction in the CMS detector. The *Event* is responsible for managing the lifetime of, and relationships between, its contents. The contents of the *Event* can include objects representing the raw detector output, reconstruction products, simulation products, and analysis objects relating to a single beam crossing or simulation thereof. The *Event* also contains metadata, describing the configuration of the software used for the reconstruction of each contained data object, and the conditions and calibration data used for such reconstruction.

The *Event* class provides the mechanism through which trigger, reconstruction, simulation, and analysis codes obtain their input. “Modules” performing discrete steps of the triggering, reconstruction, or simulation of event data communicate by obtaining the inputs from, and putting their outputs into, an *Event*.

Elements of “event data” put into the *Event* are called *EDProducts*. There is no actual class *EDProduct*; rather, *EDProduct* is a *generic programming concept*. An object of almost any type can be used as an *EDProduct*; the only *formal* requirements (enforced by the code) on the type are:

- the type must be default-constructible,
- the type must be copyable, and
- the type must be destructible.

1.3.5.2 Module types, communication

The purpose of a module is to encapsulate a unit of clearly defined event-processing functionality, in an independently testable and reusable package.

1.3.6 General Characteristics

Here are some characteristics of *Modules*:

¹In C++, a *class* bundles together some amount of data with the set of functions relevant for manipulating those data.

Modules is the generic term for all “workers” in the framework. Not all modules have the same interface.

Modules are scheduled by the *ScheduleBuilder*, and invoked by the *ScheduleExecutor*. Each *Module* instance is configured with a *ParameterSet*.

Modules must not interact directly with (*i.e.* call) other modules.

Only *Modules* are “configurable.” An internal algorithm is configured by “percolating” *ParameterSets* to the algorithm, by the *Module* that contains the algorithm. In order to provide for modular testing, which is important for quality assurance of the physics results, we require that modules communicate *only* through the *Event*, by putting *EDProducts* into the *Event*. Furthermore, we require that one may “cut” the event-processing chain between any two modules, and save the state of the event at that instant. This requires that all *EDProducts* be *persistable*.

While each *EDProduct* must be *persistable*, this does not imply each one must be persisted for every event. The event output mechanism must be capable of selective writing of *EDProduct* instances to several output streams.

Here is a (possibly non-exhaustive) list of framework module types:

- event data producers—reconstruction, and simulation
- mixing
- output
- filter
- analyzers (read-only)

Note that *input* provided by a service, not by a module.

1.3.6.1 Scheduler, Paths

We will support two different “styles” of event-processing application in the same software framework. One style of application supports *reconstruction on demand*, in the style of the previous ORCA framework. The other style is more similar to the style of the CDF and DO trigger and reconstruction frameworks. We call these styles *unscheduled* and *scheduled*.

1.3.6.2 Commonalities

For both the *unscheduled* and the *scheduled* applications, *EDProducer* instances are the objects that actually perform the task of reconstruction. An author of an *EDProducer* does not need to choose to support one or the other style of use; any *EDProducer* is able to be used in either mode.

For both styles of application, the same *EDProduct* classes are used, and the same *EDProduct* instances will be produced from identically-configured *EDProducers*.

For both styles of application, the same parameter set system is used to configure the *EDProducers*.

For both styles of application, the same input and output formats are supported.

1.3.6.3 The Unscheduled Application

In the unscheduled application, the action of requesting an *EDProduct* from the *Event* may cause the invocation of an *EDProducer*. The high-level view of the mechanism is:

1. User code requests an *EDProduct* through the `Event::get` member template, possibly specifying a selector.
2. The *Event* looks for any already-created objects of the correct type (and that match the selector, if one was provided). Such objects may be already loaded in memory, or may be retrieved from the input source.
3. If no match was found, the *Event* queries a registry of *EDProducers* to discover which ones are able to create *EDProducts* of the correct type (and which could match the provided selector, if any). If no such matches are found, the user will receive an indication that no match is available. No new libraries can be loaded at this time.
4. Any *EDProducers* found in step 3 are invoked, creating their products and entering them into the *Event*, and possibly causing a cascade of other reconstruction.
5. Any *EDProducts* generated from the *EDProducers* just invoked are returned to the user. If no appropriate producers were found, no products may be returned.

An unscheduled application is configured by specifying:

- a selection of independent top-level *EDProducts* to be written out, or
- a selection of independent high-level triggers to be run, or
- an analysis module to be run, or
- some combination of the above.

and also

- the menu of *EDProducers* that should be known to the registry of *EDProducers*.

The combination of *EDProducts* in the input source and *EDProducers* registered in the program are the only things that limit the variety of *EDProducts* than can be obtained from any *Event*.

1.3.6.4 The Scheduled Application

A scheduled application is configured by specifying a module instance path through which the event will flow. More derived or calculated products will be added to the event as it moves through the path.

The responsibility of getting the proper dependency ordering within an explicitly specified path lies with person configuring the job. However each path can be thought of independently. It is the job of the framework to optimize the schedule given a set of fully self contained reconstruction paths.

1.3.6.5 Provenance

It is critical for users to be able to unambiguously identify how each reconstruction result was produced. There are several varieties of information that constitute this identification.

Collectively, we refer to all this information as the *provenance* of the *EDProduct*. Each *EDProduct* is associated with a *Provenance* object that records this information. Where appropriate, *Provenance* objects are shared between *EDProduct* instances.

1. Module configuration

- (a) The unique identifier representing all (the names and values) of the run-time configuration parameters given to the module.
- (b) A string giving the fully-qualified class name of the module.

2. Parentage

A vector of the unique identifiers of the *EDProducts* used as inputs for this bit of reconstruction.

The identifiers are unique to the event. It is possible to maintain common identifier lists and tag those with an ID and only record.

Although a module can make use of more than one input to create its output, we make no attempt to specify the *type* of the *EDProduct* to which each of the entries in this vector refer. If such identification is needed in a particular *EDProduct*, that product can store the information in its own member data. We rejected providing a *mapping* of class name to *EDP_id* because we deemed the complexity unwarranted for the simple use to which the “parentage” information, in this general form, is put.

3. Executable configuration

- (a) A “human friendly” string called a module label, which is a unique identifier (within a job) used for *EDProducts* created by the module configured by this label. This label comes from a module configuration parameter with a fixed name. Each module has exactly one of these.

The label configuration parameter is special. Changing the label in the configuration will cause a new module to come into existence because a unique *ParameterSet* determines module instances. However, the label is not part of the permanently generated ID.

- (b) A single version number that defines the code for the entire executable. The user can obtain specific library version numbers by querying a central database, using this version number.

The value is only meaningful for tagged releases.

This number specifies which libraries were *available* when building the application; it does not indicate that *all* such libraries were used.

4. Conditions Data

An identifier representing the calibration and alignment set that was used in the construction of this *EDProduct*.

We assume here that calibration and alignment are handled in the same way and that this single, high-level identifier refers to all the calibration information used for this event. It is possible that individual calibrations (*e.g.* , silicon, calorimeter, muon) will also have IDs associated with them and that each of these will need to be recorded instead of the “set” ID.

Other conditions data IDs may also be needed here, such as geometry version or hardware configuration.

5. Job configuration

A physical process name. A job starts up in a particular context such as HLT or Reconstruction. This name identifies the process under which the job was started and is likely to be a run-time property.

All of this provenance data is distinguished from the event data because its principal home is in an ancillary database, although a copy may be readily accessible from the event data (*e.g.* , within the file that contains events).

A *Provenance* serves to collect the relevant information describing *how* a given *EDProduct* was created. Each *EDProduct* is associated (in an *Event*) with *one Provenance*.

1.4 Event Filter

Editor(s): Emilio

1.4.1 Introduction

The CMS Trigger and Data Acquisition System (TriDAS) is designed to inspect the detector information at the full crossing frequency and to select events at a maximum rate of $O(10^2)$ Hz for archiving and later offline analysis. The required rejection power of $O(10^5)$ is too large to be achieved in a single processing step, if a high efficiency is to be maintained for the physics phenomena CMS plans to study. For this reason, the full selection task is split into two steps. The first step (Level-1 Trigger) is designed to reduce the rate of events accepted for further processing to less than 100 kHz. The second step (High-Level Trigger or “HLT”) is designed to reduce this maximum Level-1 accept rate of 100 kHz to a final output rate of approximately 100 Hz. The design of the Level-1 Trigger has already been extensively documented in Volume I of the TriDAS Technical Design Report [?]. The design of the Data Acquisition System is documented in the Volume 2 [?].

The functionality of the CMS DAQ/HLT system can be summarized in three points:

- perform the readout of the front-end electronics after a Level-1 Trigger accept and assemble data from a given bunch-crossing in a single location (the memory of a computer);
- execute physics selection algorithms on the events read out, in order to accept the ones with the most interesting physics content;
- forward accepted events, as well as a small sample of the rejected ones, to online services monitoring the performance of the CMS detector
- provide the means of archiving accepted events in mass storage

1.4.1.1 DAQ Architecture Overview

The following is a schematic summary of the main functional elements as depicted in Fig. 1.4.1.1:

- Detector Front-ends: the modules that store the data from the detector front-end electronics upon the reception of a Level-1 Trigger accept signal.

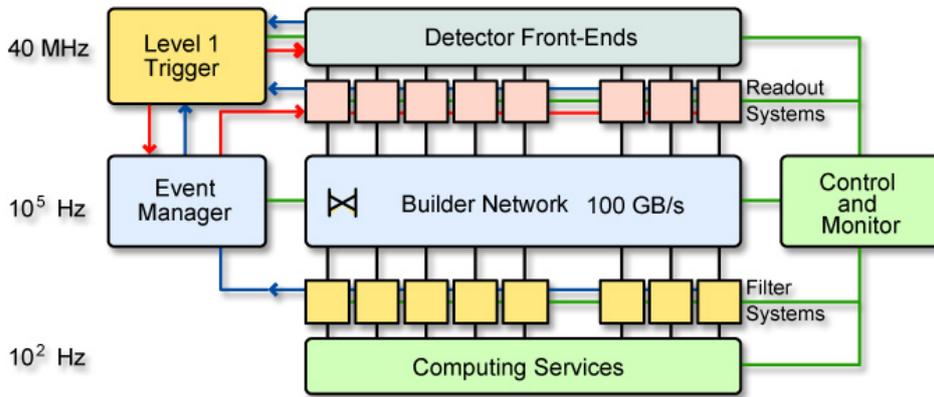


Figure 1.2: General architecture of the CMS DAQ System

- Readout Systems: the modules that read the data from the detector Front-End and store them, until they are sent to the processor which will analyse the event.
- Builder Network: the collection of networks that provide the interconnections between the Readout and the Filter Systems. It is a large switching fabric, capable of supplying 800Gb/s sustained throughput to the Filter Systems.
- Event Manager: the entity responsible for controlling the Builder Network data flow.
- Controls: all the entities responsible for the user interface and the configuration and system monitoring of the DAQ.
- HLT Systems: the ensemble of the components providing control, input data, monitoring and error detection services, and the processors executing High-Level Trigger algorithms. Approximately 500 Builder Units receive the incoming data fragments corresponding to a single event and build them into full event buffers. An appropriate number of Filter Units are connected to each Builder Unit, to provide the necessary processing power to carry out the High-Level Trigger selection.
- Services: all the processors and networks which receive or route complete or partial events, or online monitoring information, from the Filter Farm.

1.4.1.2 HLT System Requirements

The High-Level Trigger must reduce the event rate output by the Level-1 Trigger by a factor 1000 for a total output to storage of 10² Hz. At the design luminosity of the LHC this total expected to be output to mass storage, corresponds to a cross section of 10nb. Given that the $W+e+e$ production cross section alone is of this order, a significant physics selection has to take place online. It is this aspect of the HLT system that places

the most stringent requirements on the system. The main requirements on the system are thus:

- The system has to provide enough bandwidth and computing resources to minimize the dead-time at any luminosity, while maintaining the maximum possible efficiency for the discovery signals. The current goal is to have a total dead-time of less than 2%. Half of this dead-time is currently planned to be spent in the Level-1 Trigger system.
- The HLT system should tag selected events with specific trigger selection paths that were satisfied. This information can then be used by the offline system for a quick sorting of the events into physics streams.
- The system should allow for the readout, processing and storage of events that will be needed for calibration purposes.
- The HLT efficiency should not depend strongly on changes of the calibration and detector alignment constants. It must in any case be possible to validate trigger selections and compute their overall efficiency using only the data itself, with as little reference to simulation as possible.
- The system has to be flexible enough to adapt to changing run and/or fill conditions. As an example, the instantaneous luminosity is expected to drop in the course of a fill, and therefore an optimal allocation of resources might require to change trigger conditions, for instance by lowering trigger thresholds or decreasing pre-scale factors for select channels. All such changes, along with any other changes in the running conditions, must be logged.
- The system should provide enough resources to monitor the status of the CMS detector, and to provide enough information to the experimenters in case of problems.
- To maximize the efficiency of the filtering process, an event should be rejected as soon as possible. Furthermore, the system should not rely on the presence of all the information from the CMS detectors.

Given the unprecedented rate of online rejection, a most important task of the HLT is, after achieving this rejection, to provide enough information on what is rejected. It is for this reason that a major aspect of the HLT System concerns the control and monitoring of the algorithm running in the Filter Farm.

1.4.2 Filter Farm Architecture

1.4.3 Data Flow

FIXME: Describe here the data flow from raw data input into the Filter Unit to events out to the Storage Manager

1.4.3.1 Raw Data

Nicola **FIXME:** General aspects related to raw data handling (common headers/trailer, conversion, fault tolerance, PERFORMANCE)

1.4.3.2 Digi

Nicola **FIXME:** General stuff about digis. Does it really fit here ?

1.4.3.3 Output Service and Storage Manager

Jim/Emilio

1.4.4 Control Flow

Emilio/Nicola **FIXME:** The place to discuss run start/stop, condition handling, trigger counters etc.

1.4.4.1 Fault Tolerance, Exception Handling, Logging

FIXME: all that doesn't fit elsewhere ?

1.5 Detector Description

Editor(s): Mike, Frank, Oliver, Chris

FIXME: 5 pages

1.5.1 Overview

1.5.2 Geometry DB Schema

1.5.3 Geometry Model

1.5.3.1 Material

1.5.3.2 Readout Geometry

1.5.3.3 Interface between Framework and Detector Description for simulation

1.5.3.4 Interface between Framework and Detector Description for local reconstruction

1.5.3.5 Interface between Framework and Detector Description for global reconstruction

1.5.4 Unique Numbering schema

1.6 Simulation

Editor(s): Maya

FIXME: \sim 7 pages

1.6.1 Introduction

The CMS detector and physics simulation is based on the GEANT4 simulation toolkit and the CMS object-oriented framework and event model. GEANT4 provides a rich set of physics processes describing in detail electromagnetic and hadronic interactions. It also provides the tools for the implementation of the full CMS detector geometry and the interfaces required for retrieving information from particle tracking in the detectors and magnetic field. This functionality is interfaced to the CMS framework, which utilizes

the concepts of "event setup" to create the application context in terms of detector and magnetic field description, input services to pass the generated events to the simulation application, module factories to select and load physics lists and miscellaneous utilities, parameter sets for run-time configuration, and a signaling mechanism for user monitoring actions. A framework-based event producer is responsible for loading and saving the hits and digits from the sensitive detectors and the Monte Carlo truth in terms of particle tracks and vertices.

The simulation manages all CMS detectors, both central (Tracker, Calorimeters and Muon Systems) in the CMS 4 Tesla magnetic field, and forward (CASTOR calorimeter, TOTEM telescopes, Roman Pot detectors and the Zero Degree Calorimeter, ZDC), as well as several test-beam prototypes and layouts. It implements their sensitive detector behaviour, track selection mechanisms, hit collection and digitization.

The simulation has been validated by comparisons with test-beam data as well as results from its predecessor, the GEANT3-based program. It has been deployed since the 2004 CMS Data Challenge. More than ??? million events for various LHC physics channels have so far been produced.

1.6.2 Simulation Data Flow

1.6.3 Event Generators

FIXME: Parameters for pile-up and underlying event. Save detailed discussion of generators of physics processes for Vol. 2

In CMSSW, the Monte Carlo event will be contained in an OO class based on HepMC. The old HEPEVT structure is abandoned. GenEvents in HepMC are based on the natural structure of an event i.e. vertices with ingoing and outgoing particles. Iterators are implemented for easy navigation though the event.

The events will be read in to the software from a rootuple containing HepMC events. The rootuple is produced by the Generator Production Framework.

Most of the event generators that provide the collision events as input for the detector simulation are still written in FORTRAN (PYTHIA, HERWIG). Several projects to write event generators in C++ are ongoing (PYTHIA8, HERWIG++, SHERPA), and mature code is expected by 2007. The Generator Production Framework therefore has to be able to handle both FORTRAN and C++ generators

The design of the Generator Production Framework looks as follows:

- An input file contains all parameters needed to run the event generator. The generator production framework reads in the input file and passes the parameters to the generator.

- The generation step: calling FORTRAN or C++-routines.
- The conversion of HEPEVT common block to HepMC objects if needed.
- The selection of the events on HepMC level
- The persistency, i.e. writing the HepMC events in a rootuple using ROOT and SEAL.
- The output of the generator production framework is a rootuple, containing HepMC events.

This rootuple is then read in by the framework and the HepMC events are stored as EDProducts (IOMC input service).

1.6.4 Geant4 Simulation and Physics Parameters

1.6.5 Pile-up Treatment

1.6.6 Detector Digitization

The digitization step, second part of the preparation of a simulated sample, consists in the simulation of the electronic readouts used to acquire data in the DAQ system. It starts from the position and simulated energy losses in the Geant sensitive detectors, and produces an output which needs to be as close as possible to the real data from CMS, plus the additional Monte Carlo truth information available in the simulation production.

In tracking detectors (strips and pixels) the energy loss is distributed between the entry and exit point in the detector module along the path, typically of the order of $10 \mu m$, and Landau fluctuations are taken into account. The charges are drifted to the detector surface taking into account Lorentz drift, and diffused in the perpendicular plane. On the detector surface, charges corresponding to each pixel or strip are integrated, and a gaussian noise is added. Noise, if exceeding a given threshold, is also added to the other channels. Then, couplings between channels are taken into account, and conversion to digital counts is applied using the gain of the detector and the time with respect to the signal bunch crossing. If zero suppression, is required, only channels with signal exceeding a suppression threshold are saved. Additional and more specific information can be found in ???????

In ECAL digitization, The active volumes of the ECAL, for which the deposited energy is recorded (i.e. the output of the GEANT3/CMSIM step in the simulation chain), are the crystals and the silicon strips. The arrival time of hits, as well as the energy deposited, is recorded. For the crystals the variation of the light collection efficiency along the length of the crystal is simulated by multiplying the energy, as it is deposited, by a nominal

longitudinal light collection curve. This is a simple function of the distance from the crystal front-face, and is flat in the front half of the crystal, with a 5% nominal curve, as measured on production crystals, which deteriorate the energy resolution, are accounted for by a random smearing which is added later to the energy. Additional and more specific information can be found in ????????

In HCAL digitization, the simulation of the electronics for the HB/HE/HO starts with the information provided from GEANT on the energies deposited in the scintillator and their timing. The energies are converted to number of photoelectrons, and fluctuations are applied assuming a Poisson distribution. Noise is added, Gaussian in E, equal to 1.5 photoelectrons per time sample per read out depth segment, uncorrelated between time buckets (which corresponds to about 240MeV after the corrections described below). The pulses from the different energy depositions (both from the current crossing and from up to 5 previous and 3 subsequent crossings) are then added. The HF electronics uses a conventional photomultiplier tube instead of a hybrid photodiode, but nevertheless the same electronics as for the HB/HE/HO is used. The HF pulse shape is short enough to be entirely contained in one bunch crossing and is thus not affected by pile-up from previous or later bunch crossings. The magnitude of the noise used in the HF simulation is 0.125 photoelectrons, and the ADC count size is set at 0.43 photoelectrons.

In the Muon Drift Tube system, the responses of the TDCs is the output of the digitization step. Particular care is taken in simulating the behaviour of the drift cells as a function of the muon direction and impact position with respect to the sense wire, and of the residual magnetic field in the air gaps of the magnet iron yokes, where the chambers are located. The resulting drift time is smeared so as to obtain a 4 ns resolution, corresponding to an intrinsic cell resolution of about 220 μm , as measured in test-beam data. The TDC output signal for the hit reconstruction was obtained from this drift time by adding the muon time-of-flight from the collision vertex and the propagation time of the signal along the cell wire. The average muon time-of-flight from the collision vertex to a given chamber is assumed to have been subtracted at TDC level.

The digitization step of the Cathode Strip Chamber system involves simulating the responses of the ADCs and discriminators connected to the strips and wires. To create the analog signals seen by the CSC wire and strip electronics, parameterizations of the amplifier and shaper response are convoluted with the ion drift collection time. Note that the signal may contain contributions from drifting electrons due to background hits from other beam crossings. Cross-talk, both capacitive and resistive, is included in the strip signal. Each strip which satisfies the Local Charged Track (LCT) comparator logic causes the readout of a group of 16 strips in the simulation. Within such a group, noise is simulated on the empty strips that neighbour the signal strip, and remaining empty strips are suppressed. A readout dead time of 200ns is assumed. This differs slightly from the actual DAQ electronics readout, where the presence of a LCT initiates the readout of a front-end board that covers a region 16 strips wide and 6 layers deep, but the effect is expected to be negligible. Finally, the storage of the strip signals in Switched Capacitor Arrays (SCA) is simulated. The signal shape is sampled and stored at 8 times, each 50ns

apart.

The RPC response is assumed to take place within 20 ns of the passage of a charged particle through the detector with a 3ns Gaussian distributed jitter, which also accounts for the contribution from the front-end electronics and the cables to the link board. The 20 ns wide time gates were adjusted in order to accommodate triggering signals. The RPC cluster size is set to 1.5 strips.

1.6.7 Simulation Performance

1.6.7.1 The Physics of the Detector Simulation

High energy physics experiments depend critically on the accuracy of physics generators and detector simulations. Simulated data events are used for detector design optimization, calibration, object identification, and physics analysis. The size of systematic uncertainties associated with particle discoveries, mass, or cross section measurements is tightly associated with how accurately the simulations describe the actual performance of the detector in measuring electrons, photons, and hadrons. It is imperative, for the success of a HEP experiment, to understand and tune the physics of the simulation tool to agree with the data measurements. Although not always explicitly mentioned, GEANT4 [?] simulation results are compared with test beam data, wherever available, and results from the GEANT3 [?] based program.

1.6.7.2 Tracker Validation

Tracker simulation has played a key role in the development and optimization of the simulation infrastructure and the validation process. The tracker material budget, which can only be correctly estimated with a very detailed description of all active and passive detector components, directly affects the electromagnetic calorimeter physics performance and places stringent requirements on the accuracy of the detector description and geometry construction. Correct, navigable Monte Carlo truth, for correct decay tree reconstructions, as well as the proper treatment of hard electron bremsstrahlung are of vital importance in B - τ studies, in which the tracker plays a key role. With the above requirements satisfied, tracker performance has been extensively validated in terms of tracking and hit distributions for single particles, minimum bias, and physics events. (See Chapter 6.)

1.6.7.3 Electromagnetic Calorimeter Validation

Initial studies based on a comparison between a GEANT4-based simulation and test beam data provide evidence that GEANT4 gives an excellent representation of electromagnetic

showers. (See Chapter 4.) Overall ECAL performance, in terms of energy and position resolution, is dominated by effects that are not part of the shower simulation, such as electronics noise, photostatistics, longitudinal uniformity of light collection, and crystal inter-calibration. For this reason, only gross errors are identified by a comparison of energy and position resolution. The largest sensitivity is to changes or errors in the radiation and showering in the tracker material. Unfortunately the accurate simulation of this effect cannot be validated in the test beam. The shower lateral distribution, and its fluctuations from shower-to-shower is an important quantity which can be validated comparing the Monte Carlo simulations with test beam measurements. In particular, parameters sensitive to the lateral shower shape, which effects the fraction of incident energy contained in ECAL clusters, are measured in the test beam.

1.6.7.4 Hadronic Calorimeter Validation

HCAL studies on energy resolution and linearity, e/π ratio, and shower profile are instrumental in GEANT4 hadronic physics validation, in the context of the LCG simulation physics validation project. They are based on comparisons between single particle measurements in test beam experiments and GEANT4 based simulations of the associated detector setup. In 2002-2004, several HCAL test beam experiments exposed different HCAL modules, preceded by an electromagnetic calorimeter prototype, to beams of pions, electrons and muons over a large energy range. (See Chapter 5). The data were compared with GEANT4 simulations using the hadronic physics parametric (LHEP) and microscopic (QGSP) models. The pion energy resolution and response linearity as a function of incident energy derived from the simulations are in good agreement with the data measurement within the large systematic uncertainties in the latter. Transverse and longitudinal shower profiles are studied in the 1996 and 2004 test beam experiments. Pion showers predicted by GEANT4 are narrower than those predicted by GEANT3. Showers predicted by the QGSP physics list (version 2.7) are shorter than those predicted by the LHEP (version 3.6) list, with LHEP predictions being closer to those from GEANT4/Geisha.

1.6.7.5 Muon System Validation

Single muons with momenta in the 10 GeV-10 TeV range have been simulated in the CMS detector using the GEANT3 and GEANT4 packages. While both packages are in good agreement modeling ionisation, muon bremsstrahlung, e^+e^- production, and in particular, muon-nuclear interaction are significantly different, due to newer theoretical developments included in GEANT4. Multiple scattering is significantly smaller in GEANT4, in agreement with experimental results [?]. GEANT4 results also show an improvement with respect to GEANT4 in the precision of the propagation of the muons along the detector. The production threshold on secondary particles in the different regions of the detector were set to a large value to avoid the removal of tracks reaching the sensitive detectors. The

production of hits in the simulation of the muon system was tested by comparing the Monte Carlo predictions with test beam data. (See Chapter 3). The test beam experiment consisted of two muons chambers with and without an iron slab in between them, to investigate the effect of the muon showers in the passive material. The analysis, based on muons in the 50-300 GeV p_T range, show that GEANT4 slightly underestimates soft delta ray production in cell volumes, while hard delta rays and electromagnetic showers are correctly modeled. In spite of this discrepancy, local track reconstruction efficiency and resolution is well reproduced by the simulation.

1.6.7.6 Forward Detectors

The forward detectors, such as the CASTOR and ZDC calorimeters, and the Totem telescopes are also incorporated to the simulation framework. They are essential tools for the diffractive and heavy ion programs. (See Chapter 7). For example, the ZDC is a Cerenkov detector designed to collect any remaining neutral fragments of the colliding nuclei and may be used as a measure of the collision centrality. In pp collisions, the ZDC may be incorporated in the study of forward physics and photon production. Simulation studies are underway to study issues such as energy resolution and energy leakage. Test beam data available in October or November 2005 will allow for more systematic validation of the simulation results. Current validation efforts involve implementation of a RHIC-design ZDC in OSCAR to take advantage of existing test beam data.

1.6.7.7 Parameterized Showers

The detailed simulation of electromagnetic showers is computationally intensive. A parametrization of the spatial energy distribution of an electromagnetic shower, based on probability density functions, allows to speed up the process without compromising the simulation accuracy. A shower parameterization model called Gflash, based on three probability density functions, was developed and used by the H1 experiment [?]. As part of the GEANT4 software distribution, Gflash is available in OSCAR, the CMS detector simulation package. In CMS, GFlash is used to parameterize electrons and positrons in the barrel and endcap electromagnetic calorimeter. Comparisons between the GFlash based and the full OSCAR simulation of the energy depositions in the central crystal, and 3x3, 5x5 crystal matrices show good agreement to the $> 1\%$ level, as illustrated in Fig. 1.6.7.7. Figures 1.4- 1.5 show that the transverse and longitudinal shower profiles are also well modeled by Gflash to within 1-3%. The Gflash shower parameterizations allow a significant time performance gain in the simulation, with speed increases in the range of a factor of 3-10. The gain in speed depends on the event type, the particle energy and the detector η region. For instance, a single electron or photon with an energy of 100 GeV in the ECAL barrel is simulated 10 times faster using Gflash. For a large extra dimensions full signal event, $pp \rightarrow \gamma + G$, with a single photon above 1000 GeV, the gain in speed is a factor of 4.

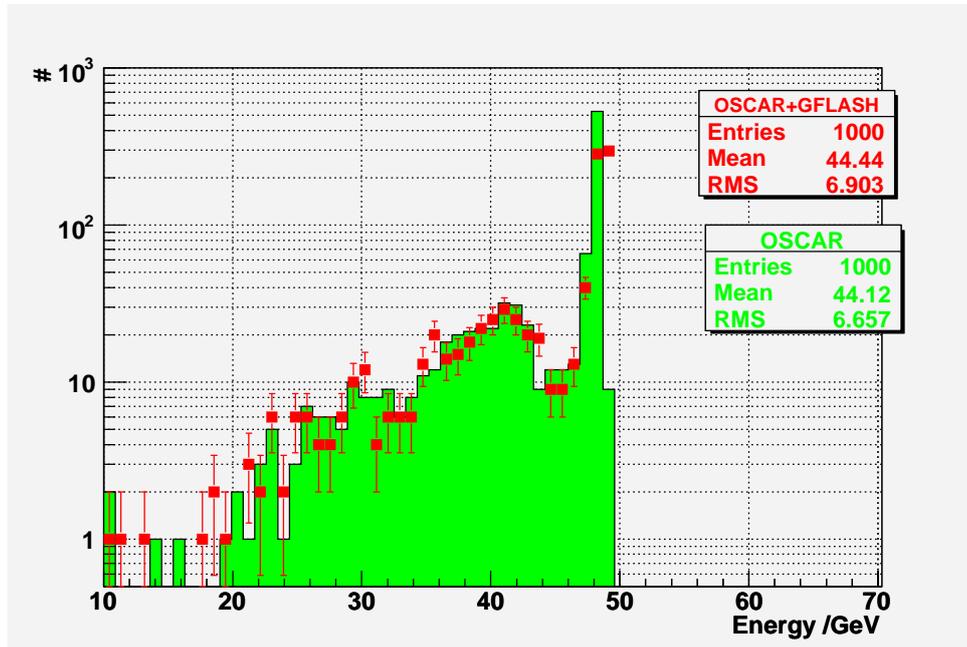


Figure 1.3: Energy depositions in a 5x5 crystal matrix for 50 GeV electrons. The histogram corresponds to the full GEANT4 simulation and the red markers to the shower parameterization.

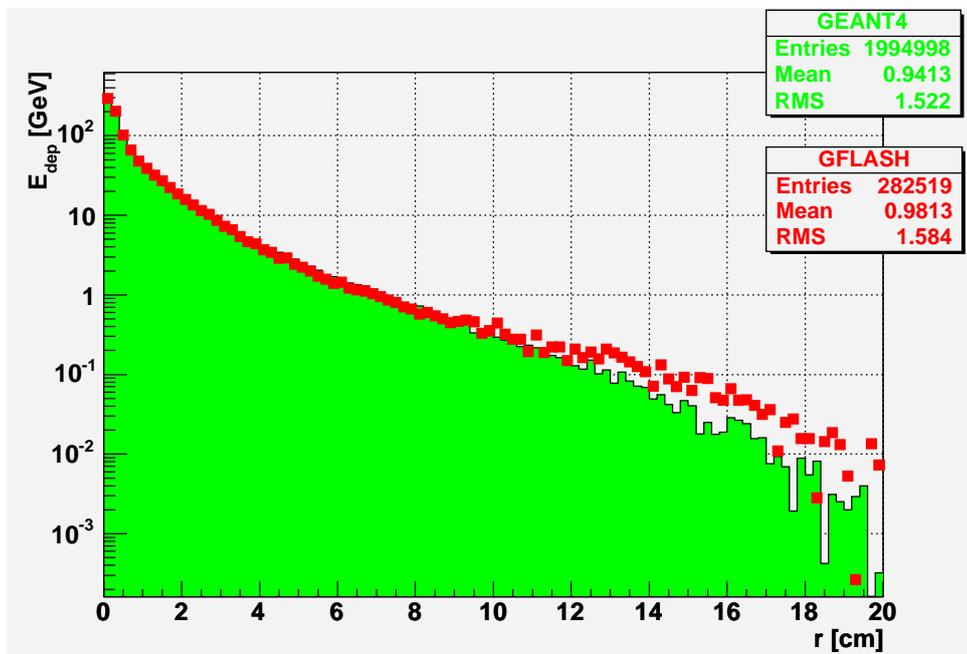


Figure 1.4: Transverse shower profiles for 50 GeV photons.

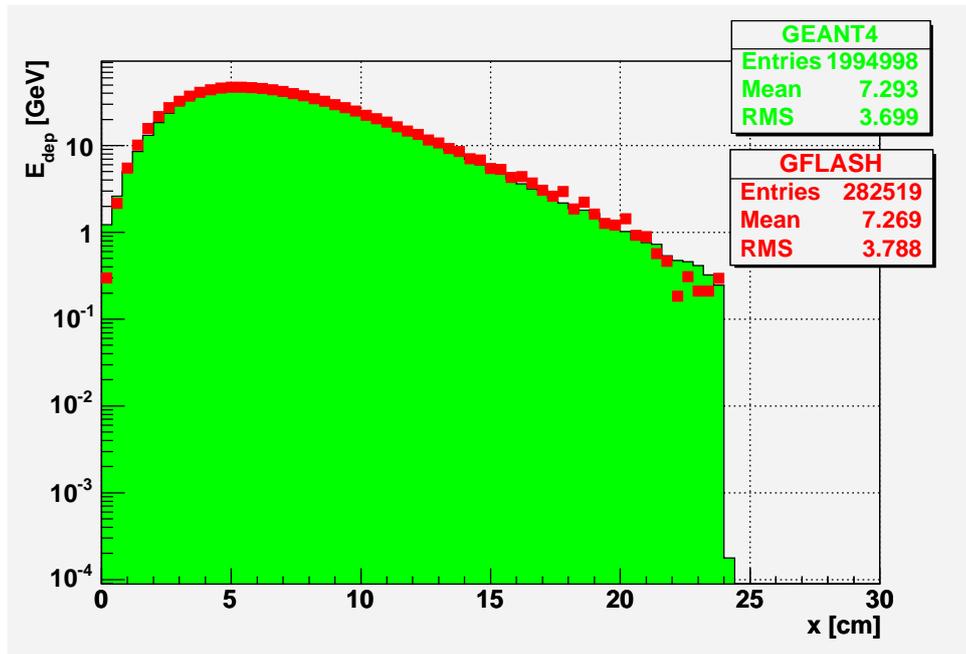


Figure 1.5: Longitudinal shower profiles for 50 GeV photons.

1.7 Fast Simulation

Editor(s): Patrick Janot

Contributor(s): Salavat Abdullin, Florian Beaudette, Patrick Janot, Andrea Perrotta

1.7.1 Introduction

A framework for fast simulation of particle interactions in the CMS detector, called FAMOS, has been recently developed, and is intended to be used for most physics analyses, in view of the Volume 2 of this Physics TDR and beyond. It is an object-oriented system for which C++ has been chosen as programming language. The acronym FAMOS stands for FAst MOnte-Carlo Simulation. As it is a work in progress, only the current status and performance are described in this section.

The input of FAMOS is a list of particles (originating from an event generator or a simple particle gun) characterized by their momentum and origin vertex, with mother and daughter relationships to follow the various decay chains in the event. Upon user request, each of the (quasi)-stable particles in this list is then propagated in the CMS magnetic field to the different layers of the various CMS subdetectors, which it may interact with. While propagating, these quasi-stable particles are also allowed to decay according to

their known branching fractions and decay kinematics. The particles resulting from the interactions with the detector layers or from the decays in flight are added to the original list, and propagated/decayed in the same way. Events from pileup interactions in the same bunch crossing as the original event are read from pre-generated files, added to the list according to a Poisson distribution with a user-defined average, and follow the same treatment.

The interactions simulated in FAMOS are *(i)* electron Bremsstrahlung; *(ii)* photon conversion; *(iii)* charged particles energy loss by ionization; *(iv)* charged particle multiple scattering; and *(v)* electron, photon and hadron showering. The first four are applied to particles traversing the thin layers of the tracker (Section 1.7.2), while the latter is parametrized in the electromagnetic (Section ??) and hadron (Section ??) calorimeters. The plans are to simulate all five interactions together with synchrotron radiation for the muon propagation through the tracker, calorimeters and muon chambers. The current muon simulation is, however, mostly based on a parametrization of resolutions and efficiencies, as is explained in Section ??.

As output, FAMOS delivers series of “high-level objects”, such as reconstructed hits for charged particles in the tracker layers, energy deposits in calorimeter cells, which can then immediately be used as inputs of the same “higher-level algorithms” (track fitting, calorimeter clustering, b tagging, electron identification, jet reconstruction and calibration, trigger algorithms, ...) as in the full reconstruction and analysis package, ORCA. This parallelism between the fast simulation and the complete reconstruction allows comparisons between fast and full simulations and subsequent tuning in a straightforward manner, with the use of identical analysis programs. It also allows the development of new reconstruction algorithms with the fast simulation, for later use in the complete reconstruction package.

Last but not least, the computer time needed to simulate an event in FAMOS is about three orders of magnitude smaller than that needed in the full chain, for a level of agreement aimed at the percent level or below. At the time of writing, the simulation of one LHC event in FAMOS in its highest level of sophistication takes a second or thereabout on a 1.8 GHz computer, with expected optimizations still to come.

1.7.2 Simulation of the Tracker Response

A simplified version of the tracker geometry, but deemed adequate for the required level of accuracy, is used. It is made of over 30 thin nested cylinders representing the sensitive layers of, from inside to outside, the pixel detector (three barrel layers and two endcap disks), the four tracker inner barrel layers, the three tracker inner disks, the six tracker outer barrel layers and the nine tracker endcap disks, interleaved with non-instrumented cylinders with dead material (cables, support, ...). The material, assumed to be pure Silicon, is also assumed to be uniformly distributed over each cylinder barrel (respectively, endcap). A transverse view of this simplified geometry is shown in Fig. ??a, in