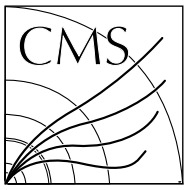
Available on CMS information server

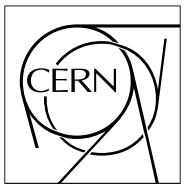
CMS NOTE 1999/072



The Compact Muon Solenoid Experiment

CMS Note

Mailing address: CMS CERN, CH-1211 GENEVA 23, Switzerland



December 10, 1999

CMS simulation software using *Geant4*

S. Banerjee[a)], I. González[b)], V. Lefébure[c)]

Abstract

Geant4 has been chosen as the basic toolkit for carrying out detector simulation work in CMS. We have developed a prototype of what could be a framework to build the CMS geometry and to construct the corresponding *Geant4* physical volumes from that. We have used it to simulate all the components of the CMS detector and to produce transient hits. Here we describe the design and the motivations for specific choices.

[a)] Tata Institute of Fundamental Research - EHEP, Mumbai

[b)] Instituto de Física de Cantabria, Santander, Spain

[c)] CERN, European Laboratory for Particle Physics, Geneva

1 Introduction

Geant4[1] is a toolkit for detailed as well as for fast Monte Carlo simulation of High Energy Physics experiments. This package will substitute its predecessor *Geant3*[2] as the basis for CMS[3] simulation in an Object-Oriented world. Enhanced capabilities and performance are expected from this new package (*Geant4*). Although *Geant4* is based on the experience acquired for *Geant3*, it completely remodels the physics processes and improves the precision in prediction. An adequate implementation of the CMS geometry and the ability to produce hits are the first steps towards a complete simulation program. We have realised a first iteration in that direction and our work lead us to the definition of the model described in this note.

During the coding and debugging process, we have gained some experience both with *Geant4* and with CMS itself. This was, in fact, the main aim of this first prototype. Therefore, this is by no means a final framework for CMS detector simulation. It is expected to evolve during the following months and years to become a full simulation program, with higher functionality, flexibility and extensibility (see [4]).

The prototype model is described in section 2. Sections 3 to 5 deal with the different categories in this model. More information can be found in [5, 6, 7].

All the class and category diagrams are written in the Unified Modelling Language (UML) [8] and were produced with Rational Rose [9].

2 The Model

The physical division of CMS into different sub-detector components is well defined. However, software cannot follow this approach, and different categories should conform to the conceptual responsibilities of the different classes. Eight main categories have been identified: CMS geometry description, *Geant4* geometry builders, CMS and *Geant4* hits, common utilities, persistency of geometry , *Geant4* user actions and graphical user interface. Within the specific geometry category, the subdetector component structure could be followed and implemented.

From the point of view of the dependencies among the different categories, a special effort has been made to avoid any knowledge of *Geant4* classes from the CMS specific ones, like the one dealing with CMS geometry description. Therefore, the CMS core categories and the model used for building the simulation geometry could be easily reused in other fields where CMS geometry is required, i.e. event reconstruction, visualisation, etc. This constraint implies a complex implementation to avoid code duplication and to simplify class interfaces.

The different categories used in the current model are shown in figure 1:

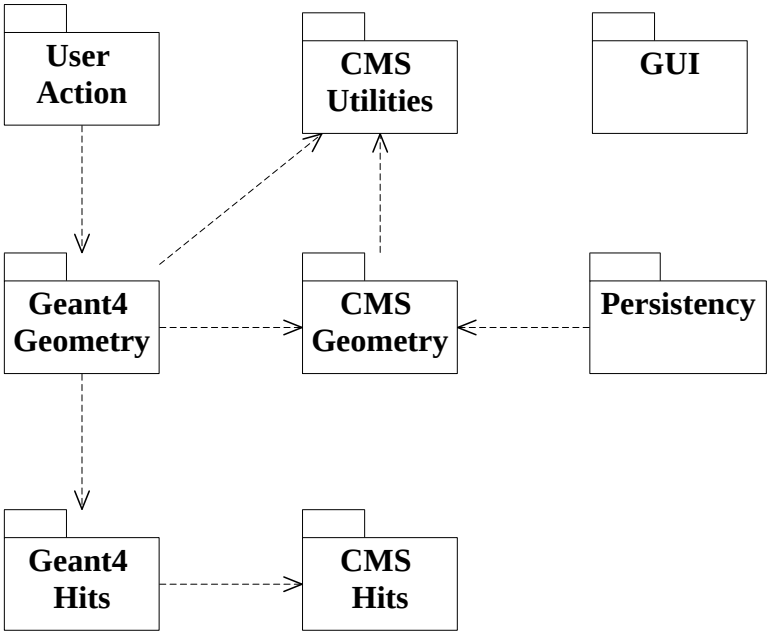


Figure 1: *UML category diagram of the current model of CMS detector description*

- **CMS Geometry Description**: It includes the transient version of the factory classes that provide a description of the CMS detector components.
- ***Geant4* Geometry**: This category contains the factory classes which create a transient instantiation of the detector setup within the *Geant4* framework.
- **CMS Persistent Geometry**: The persistent description of the CMS detector (persistency has been achieved using ODBMS technology with Objectivity DB [10] and HEPODBMS [11]) has been put into this category.
- **CMS Utilities**: It groups general classes to specify materials, rotation matrices, magnetic field, interface to ntuple and histograming facilities and a few other utilities like string or stream manipulation.
- ***Geant4* User Action**: Here are the CMS user action classes that allow the user to plug-in actions within the *Geant4* framework.
- **CMS Hits**: This category contains the CMS hit classes and related classes such as the definition of the numbering scheme of the sensitive detector units.
- ***Geant4* Hits**: It includes all classes needed by *Geant4* to produce hits.
- **GUI**: This category contains the only classes written in Java: they provide a Graphical User Interface.

3 Geometry Factories

Though the CMS detector will have a unique geometry, the detector geometry will have different views in different types of applications. For instance, a reconstruction application may need a very accurate description of the sensitive components and an average description of the inactive components, while the simulation program would require a detailed description of the distribution of all the materials in the detector. The *CMS Geometry Description* category maps the actual geometry of CMS in a transient model, and at the same time it also provides ways in which the different views can be plugged in. The classes in the *Geant4 Geometry* category, are, currently, the only ones, in this prototype, that construct a view of the CMS geometry based on the CMS factory classes. The reconstruction project, ORCA [12] is currently using a geometry model based on the *Geant3* description in *CMSIM* [13]. The classes in the *CMS Persistent Geometry* category is the persistent representation of the actual CMS geometry. The relation between the transient and persistent part of the geometry, as well as the connection between the actual geometry and the *Geant4* view is shown in figure 2. A polymorphic approach has been taken for the three categories.

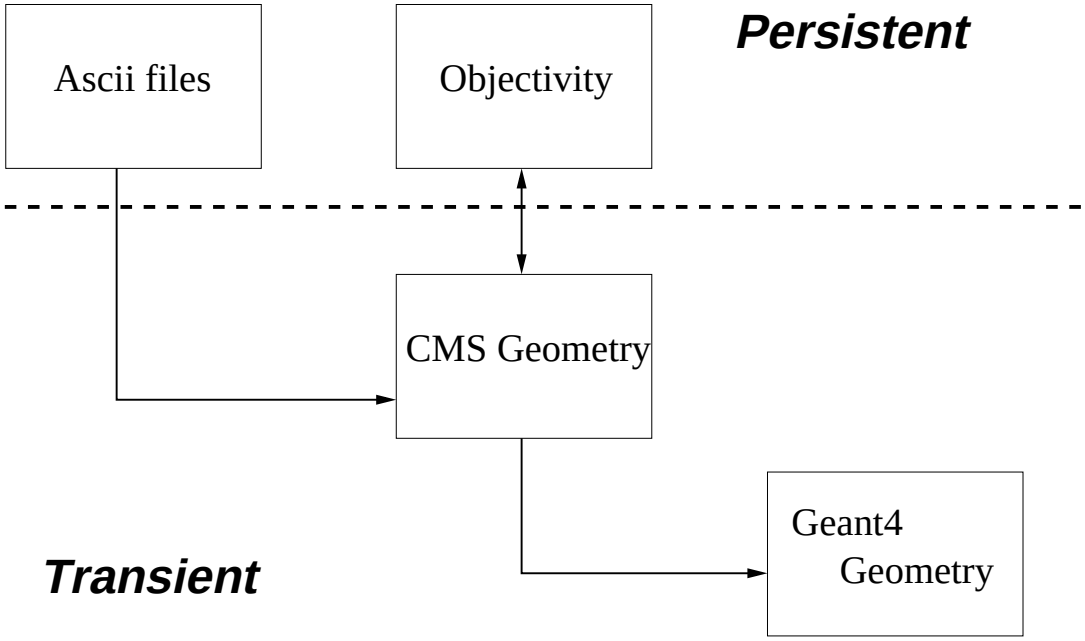


Figure 2: *Relation between the Persistent and Transient version of the CMS Geometry, and the building of the Geant4 view*

3.1 CMS Geometry Description

The CMS geometry is built within the *CMS Geometry Description* category by making each sub-detector (down to a certain level) inherit from the common base class `CMSDetector` (see figure 3). The inheriting classes will hold basically two types of information:

- Dimensional parameters, number of components, names and materials needed to fully describe the corresponding sub-detector. No special format has been imposed as yet. Each sub-detector code is separately documented and appropriately implemented.
- The geometry hierarchy. This means each sub-detector knows exactly to whom it belongs (its mother) and what it consists of (its daughters). A common way of storing, retrieving and modifying this hierarchy is provided in the base class.

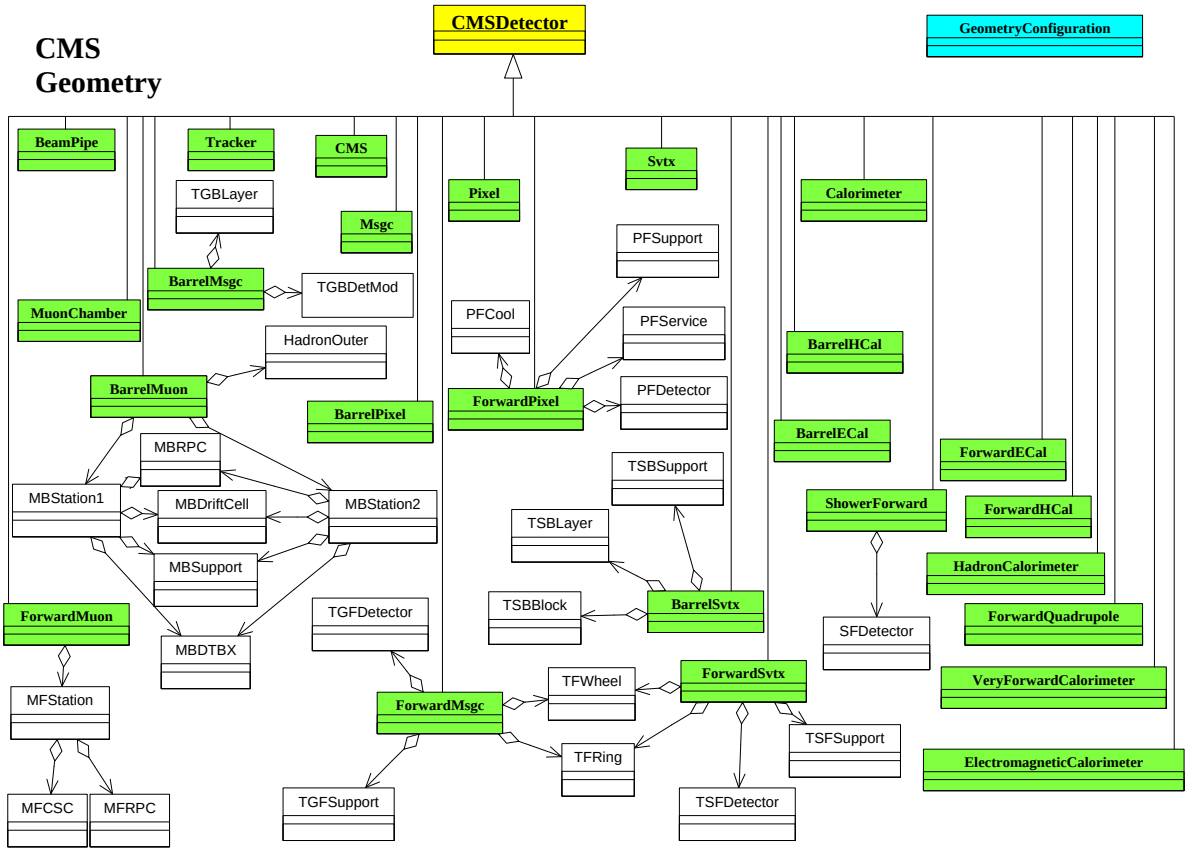


Figure 3: *Class Diagram for the CMS geometry description category. The base class* `CMSDetector` *is printed in yellow, while the main sub-detectors classes are in green. The class in blue takes care of the configuration of the geometry.*

3.2 *Geant4* Geometry

On the *Geant4* side of the geometry model, all the geometry factories are derived from the base class `G4Able`. Nevertheless, a more complex structure arises from the fact that double inheritance is used, so that each *Geant4* geometry factory class inherits also from its counterpart in the CMS specific hierarchy. `G4Able` maps the features needed by *Geant4* to build the geometry. The design of the `G4Able` class helps a modular approach and easy interchanging at the level of sub-detectors, allowing an easy transition from the simulation of the entire CMS detector to that of just a part of it, or to a test beam geometry. The sensitive property of each single detector as well as its visualisation attributes have been incorporated into the design, so that this can be selected at run time. Different parts of the sub-detectors can be classified as sensitive, electronics, support, cable, absorber, pseudo-volumes, or other services, and the visibility or colour of each group may be easily modified. An example of the detectors coming out from *Geant4* visualisation is shown in figure 4.

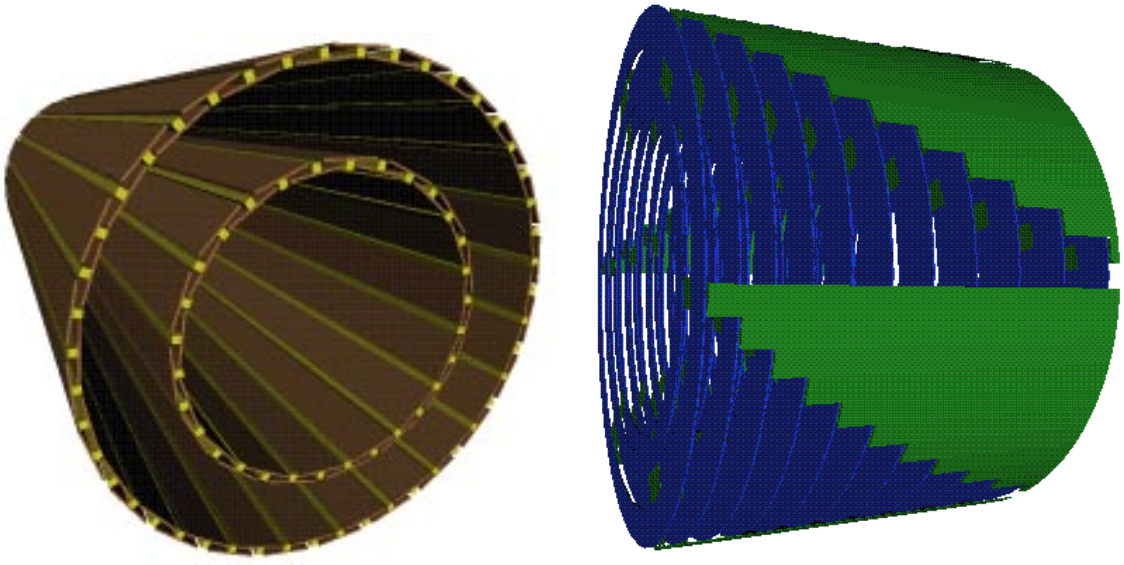


Figure 4: *Barrel Pixel (left) and Forward MSGC (right) pictures as described in the simulation application.*

3.3 CMS Persistent Geometry

The base class for the *CMS Persistent Geometry* category is called `PCMSDetector`. This class abstracts the detector information such as detector name, associations, and files from which it was created. The knowledge and handling of the federated data base is the responsibility of a dedicated persistency management class.

4 Hits

In the *Geant4* framework, an object of class `G4VSensitiveDetector` has to be assigned to the `G4LogicalVolume` of a sensitive `G4VPhysicalVolume`. That class has the responsibility to produce transient hit objects of class `G4VHit`.

The different CMS sub-detectors produce hits which need different types of information in order to be compatible with the ORCA requirements. Since they could be grouped into Tracker-like or Calorimeter-like hit objects, two classes have been defined in the current framework: `CMSTrackerHit` and `CMSCaloHit`. The interface required by *Geant4* is implemented into `G4TrackerHit` and `G4CaloHit` classes which doubly inherit from `G4VHit` and the corresponding CMS hit class.

Also two specialisations of the `G4VSensitiveDetector` class are used: `G4TrackerSD` for Tracker-like detectors and `G4CaloSD` for Calorimeters. They require the definition of a concrete class derived from the pure virtual base class `VDetectorOrganization` used by the Sensitive Detector to determine a unique identification of the sensitive units.

A more detailed description of hit and sensitive detector classes can be found in [6].

5 User Interactions

The software has been designed to be flexible at all levels. At the geometry level, the user can choose which subsystem of the detector setup should be simulated and can activate or deactivate the sensitive parts subsystem by subsystem. In addition it inherits the flexibility of *Geant4* at the level of the physics to be simulated (different types or models of simulation of electromagnetic and hadronic processes, ...), and in the management of user-actions.

5.1 Detector construction:

The class `CMSDetectorConstruction`, inheriting from `G4VUserDetectorConstruction`, provides the method to return the pointer to the physical volume of the top level mother volume. The possibility of selecting a specific detector at compilation time is provided by the use of CPP switches (as described in appendix D of [5]).

5.2 Physics list:

The class `CMSPhysicsList` publicly inherits from `G4VUserPhysicsList` and implements the virtual methods to construct particles, processes and build tables of cross-sections and energy losses depending on the cut parameters chosen for the simulation run.

5.3 Primary Generator Action:

The class `CMSPrimaryGeneratorAction` publicly inherits from `G4VUserPrimaryGeneratorAction`. It allows the user to choose among three alternatives for the generation of primary particles:

- a single particle generated at a given fixed point and moving along a specified direction. All parameters (particle type, energy, origin, direction) can be defined by the user.
- a single particle with its direction randomly distributed within a user-defined solid angle.
- HEP (e.g. PYTHIA) events as read from an ASCII file.

5.4 Run, Event, Tracking and Stepping Actions:

A complete description of the User Action mechanism can be found in [7].

The classes `CMSRunAction`, `CMSEventAction`, `CMSTrackingAction`, and `CMSSteppingAction` publicly inherit respectively from `G4VUserRunAction`, `G4UserEventAction`, `G4UserTrackingAction` and `G4UserSteppingAction`.

These classes should not be modified by the user. They provide public methods allowing to plug-in user-defined actions. The user has to create its own action classes by inheriting from the interface classes `CMSRunActionUnit`, `CMSEventActionUnit`, `CMSTrackingActionUnit`, or `CMSSteppingActionUnit`.

A few user actions already exist:

- `CaloEndOfEventAction` and `TrackerEndOfEventAction`: it prints the information concerning energy deposition in all Calorimeter and Tracker/Muon parts respectively, at the end of each event.
- `DrawTrajectoriesEventAction`: at the end of each event, it prints on the screen the number of trajectories stored in the current event and draws them.
- `DrawSteppingAction`: it draws the trajectory segments at each step of the tracking.

5.5 Graphical User Interface

A Java Graphical User Interface (GUI) is provided for the definition of the geometry to be simulated. Sensitivity and visualisation of each subsystem can be switch on or off using this GUI. These actions are propagated to all subcomponents of the subsystem. A macro file can be created, edited, or simply selected from the GUI. It provides also the possibility to redirect the output to a given file. A view of the GUI can be seen in figure 5.

6 CMS Utilities

Several utility methods together with classes to handle materials, rotation matrices, magnetic field have been provided. One can group them into five different domains:

6.1 Materials:

All materials required to define geometry of the CMS detector are kept in a database and a set of classes are provided to instantiate the specific materials (as `G4Material`) required in the specific application of geometry definition.

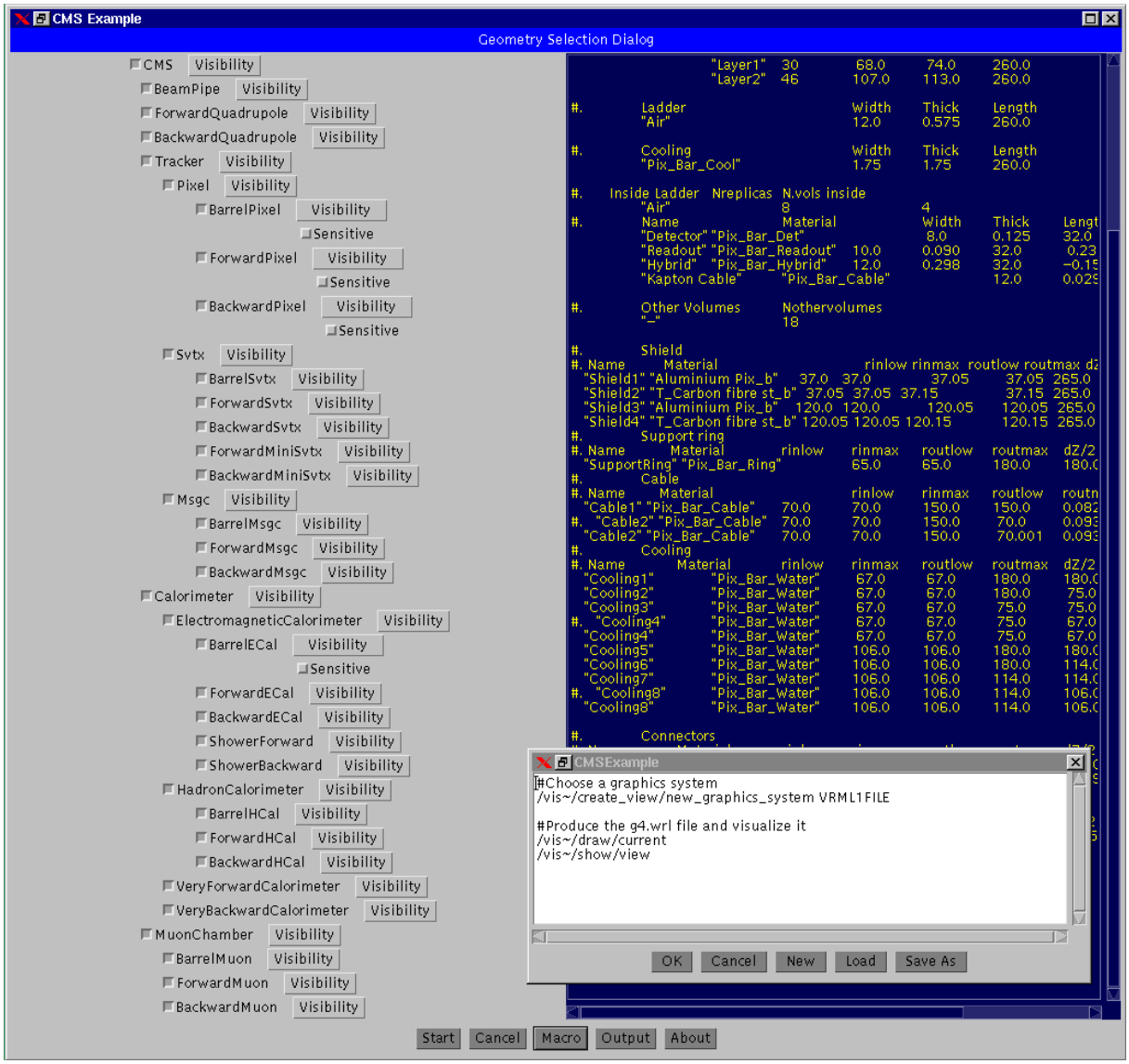


Figure 5: *An example of the use of the Graphical User Interface*

6.2 Rotation Matrices:

It provides facilities to define rotation matrices through the direction cosines. Some standard set of rotation matrices, most commonly used, are saved in a database and the required ones can be instantiated on demand.

6.3 Magnetic Field:

A three dimensional description of the 2-D magnetic field of the CMS detector is specified and can be activated in the context of tracking inside the CMS detector.

6.4 Histograming and Ntuples:

For the use of the familiar HBOOK histograms and ntuples [14], a C++ interface to the HBOOK facility is provided.

6.5 Other Utilities:

Some interesting string and stream manipulation functions are provided. The description of these utilities can be found in [5].

7 Conclusions

The first iteration of a complete simulation of the CMS detector with the Geant4 toolkit has been an excellent and fruitful exercise. The objective of building a first software prototype for CMS simulation in an Object-Oriented world using *Geant4* has been achieved easily in most of the areas. The flexibility of C++ and *Geant4*, and the power of object-oriented programming lead us to build code which is reusable, flexible and extensible. We managed to decouple the description of the CMS geometry and hits from the different possible views, and we implemented that of a simulation toolkit, in this case *Geant4*. Many individual programs with different purposes have been built using the framework described in this paper: full CMS geometry, sub-detector debugging, test-beam simulation, etc. The knowledge and experience acquired provide a solid basis for the realisation of a functional prototype which is the next step towards a final CMS simulation program [4].

8 Acknowledgements

We would like to thank P. Arce, J. Klem, M. Kossov and A. Nikitenko for offering their expertise in particular sub-detectors by contributing substantially to the implementation of their geometry description and allowing us to complete the job.

We would also like to gratefully acknowledge the *Geant4* team, and in particular J. Apostolakis, S. Giani, H-P. Wellisch, for all their quick and accurate answers to our questions and problems and for their advice which helped us to improve our model.

We should not forget to include in this section the CMS collaborators which somehow contributed with their suggestions, thoughts and interest in the former CMS *Geant4* User Club and in the new OSCAR project.

References

[1] See *http://wwwinfo.cern.ch/asd/geant/geant4.html*

[2] **CERN Program Library W5013**, Application Software Group, CN Div., *"GEANT Detector Description and Simulation Tool (Version 3.21)"*

[3] **CERN/LHCC 94-38, LHCC/P1**, *"CMS Technical Proposal"*.

[4] **CMS Internal Note CMS/IN 99/36**, M.Schröder, *"CMS Detector Simulation Project OSCAR"*

[5] **CMS Internal Note CMS/IN 99/26**, P. Arce, S. Banerjee, I. González, J. Klem, A. Nikitenko, *"Detector Description of CMS using Geant4"*

[6] **CMS Internal Note CMS/IN 99/54**, V.Lefébure, *"CMS simulation software using* Geant4*: Hits and Sensitive Detectors"*

[7] **CMS Internal Note CMS/IN 99/53**, V.Lefébure, *"CMS simulation software using* Geant4*: User Actions"*

[8] See *http://www.rational.com/uml/*

[9] See *http://www.cern.ch/PTTOOLS/Rose/*

[10] See *http://www.objectivity.com*

[11] See *http://wwwinfo.cern.ch/asd/lhc++/HepODBMS/user-guide/ho.html*

[12] **CMS Internal Note CMS/IN 99/35**, D.Stickland, *"CMS Reconstruction Software: The ORCA project"*

[13] See *http://cmsdoc.cern.ch/cmsim/cmsim.html*, *"CMSIM User's Manual and Reference Guide"*.

[14] **CERN Program Library Y250 (1994)**, *"HBOOK Reference Manual (Version 4.22)"*.