INTRODUCTION TO GEANT4

Makoto Asai
Hiroshima Institute of Technology, Hiroshima, Japan
(Geant4 Collaboration)

Abstract
Geant4 is one of the first successful attempts to re-design a major package of CERN software for the next generation of HEP experiments using an Object-Oriented environment. It is being developed and maintained by more than 100 colleagues of all over the world with adoption of the most recent software engineering methodologies, choice of Object-Orientation and C++. In this paper, emphasis is paid on how these modern techniques especially Object-Orientation affect on the design, implementation and maintenance of Geant4.

1. INTRODUCTION

1.1 Geant4 collaboration

Geant4 [1] is the successor of GEANT3, which is well-known as the world-standard toolkit for HEP detector simulation. Geant4 is one of the first successful attempts to re-design a major package of CERN software for the next generation of HEP experiments using an Object-Oriented environment. A wide variety of requirements also come from heavy ion physics, CP violation physics, cosmic ray physics, medical applications and space science applications. In order to meet such requirements, a large degree of functionality and flexibility are provided. G4 is not only for HEP but goes well beyond that.

At the beginning of 1990's, GEANT3 faced on two major difficulties on its maintenance and further development.

i) Because of too complex structure driven by too many historical reasons, it became impossible to add a new feature or to hunt a bug. This indicated the limitation of FORTRAN and/or old style of programming.

ii) Shortage of manpower at CERN became serious. This indicated the limitation of "central center" supports.

To solve these difficulties, a new R&D project (CERN RD44 [2]) was started in December 1994. This R&D project is made of a world-wide collaboration and decided the adoption of the most recent software engineering methodologies and choice of Object-orientation and C++.

This R&D project was successfully terminated in December 1998 with the first production release of Geant4. Now, the Geant4 collaboration becomes MoU-based collaboration, which is maintaining its code, documents and examples, at the same moment of upgrading its functionalities. The collaboration will continue to maintain and upgrade Geant4 for decades, e.g. lifetime of LHC.

1.2 Performance? Flexibility? Usability?

We believe that Geant4 is a fundamental test of the suitability of the object-oriented approach and C++ language for software in HEP, where performance is an important issue. As a consequence, Geant4 releases should be regularly monitored against the performance provided by Geant3 at comparable physics accuracy. For the case of geometrical navigation, Geant4 automatically optimizes the user's geometrical description, while the user had to implement his/her geometry very carefully to

exploit the full performance of Geant3 navigation. And Geant4 provides faster navigation than optimized Geant3 descriptions. For the case of electro-magnetic physics in a simple sampling calorimeter, we confirmed Geant4 is 3 times faster when using the same cuts (in the sensitive material) as GEANT3. And more than a factor 10 faster when seeking the best performance in Geant4 that maintains constant the quality of the physics results as Geant3. Geant4 is faster than GEANT3 in all aspects, when its power and features are well exploited.

The flexibility of Geant4 could be emphasized in many aspects. For example in physics processes, much wider coverage of physics comes from mixture of theory-driven, cross-section tables, and empirical formulae. Thanks to polymorphism mechanism, both cross-sections and models can be combined in arbitrary manners into one particular process. We have wide variety of processes for various kinds of particles such as slow neutron, ultra-high energy muon, optical photon, parton string models, shower parameterization, event biasing technique, etc., and new areas are still coming. Also thanks to the abstraction and polymorphism, we have many types of geometrical descriptions such as CSG, BREP and Boolean, and all the geometrical descriptions are STEP compliant. Because events and tracks are treated as class objects, overlapping events, suspension of slow looping tracks and postponing these tracks to next event, and priority control of tracks in the stacks could be realized without losing any performance overhead. We must stress that everything in Geant4 is open to the user so that he/she can choose physics processes and models, integrator in magnetic/electric field, GUI and visualization technologies, and histogramming and persistency mechanism according to his/her environment or preference.

In terms of usability, User Requirements Document (URD) states many different use-cases from various fields. Thanks to the inheritance mechanism, the user can derive his/her own classes easily. Many abstract layers and default behaviors are provided at the same time. Many reusable examples and documents are provided and are still continuously evolving with the user's contribution. Geant4 had already started to be used by various scientific fields well beyond HEP.

2. OBJECT-ORIENTED ANALYSIS AND DESIGN ADOPTED IN GEANT4

2.1 User Requirements Document

As mentioned in the previous chapter, Geant4 has to satisfy a wide variety of requirements. At the beginning of the development of Geant4, requirements and use-cases were gathered from contributors who belonged to wide variety of scientific fields. These requirements and use-cases were studied one by one, categorized and collected into Geant4 User Requirements Document (URD) [3]. All the top level designs of Geant4 based on URD.

URD is the base of Geant4. It has been being maintained through the life of Geant4. Once a new requirement comes and it is recognized as reasonable, URD is updated to get it in at the same moment of its implementation. Global design and its implementation are regularly checked for their consistencies with respect to URD.

2.2 Category Design

Figure 1 shows the top level categories of Geant4. Dashed allows show the dependencies between categories. With respect to the URD, 17 major categories were found by Object-oriented analysis and design. The most important issue of designing top level categories is establishing well-defined category boundaries with respect to the mandates of each category, and defining clear "use relations" between categories with avoiding loop dependency. For example, "Processes" category uses "Particle" via "Track", but "Material" category should not know about "Particle".

2.3 Basic concepts in Geant4

Geant4 is the Object-Oriented toolkit which provides functionalities required for simulations in HEP and other fields. Simulation is a "virtual reality". Simulation is used both to help designing detectors

during R&D phase and understanding the response of the detector for the physics studies. To create such virtual reality we need to model the abstract behaviors of the particle-matter interactions, geometry and materials in order to propagate elementary particles into the detector. We need also to describe the sensitivity of the detector for generating raw data. In this section, some basic concepts used in Geant4 are explained. These concepts are quite important for understanding the structure and the behavior of Geant4.

2.3.1 Run

As an analogy of the real experiment, a run of Geant4 starts with "Beam On". Within a run, the user cannot change neither detector geometry nor settings of physics processes. This means detector is inaccessible during a run. Conceptually, a run is a collection of events which share the same detector conditions.

2.3.2 Event

At beginning of processing, an event contains primary particles. These primaries are pushed into a stack. When the stack becomes empty, processing of an event is over. G4Event class represents an event. It has following objects at the end of its processing.

- List of primary vertexes and particles
- Trajectory collection (optional)
- Hits collections
- Digits collections (optional)

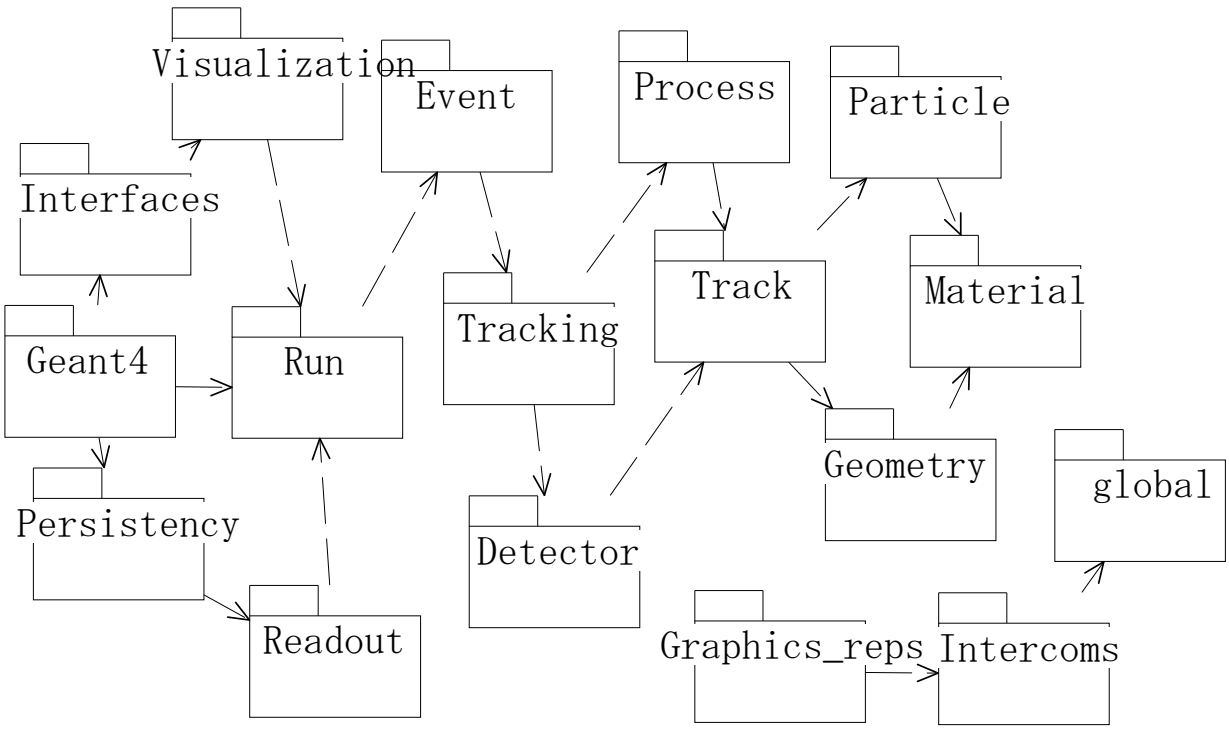

Fig.1 Category diagram of Geant4.

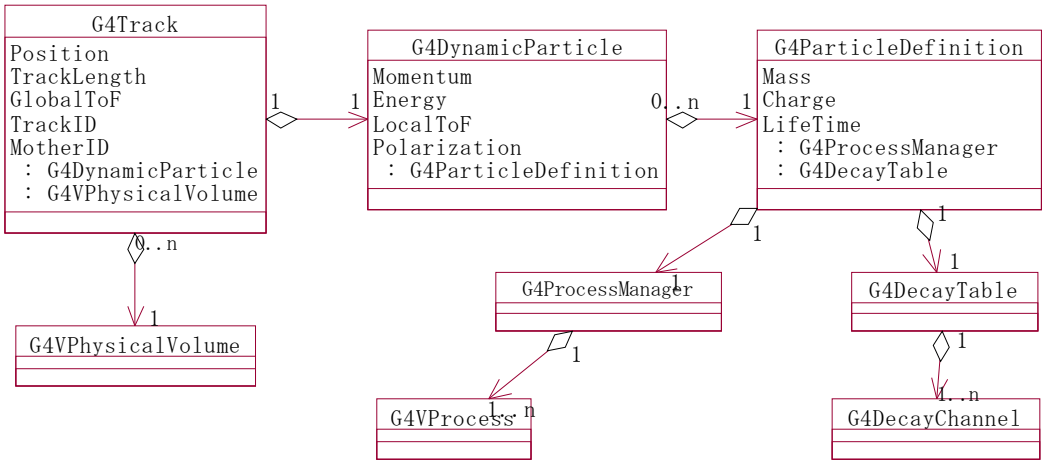

Figure 2. Classes which represent a track

2.3.3 Track

Track is a snapshot of a particle. Step is "delta" information to a track. Thus, track is not a collection of steps. Track is deleted when

- it goes out of the world volume,
- it disappears (e.g. decay),
- it goes down to zero kinetic energy and no "at rest" additional process is required,
- the user decides to kill it.

As shown in Figure 2, a track in Geant4 is represented by three layers of class objects. First two objects of G4Track and G4DynamicParticle classes are unique for each track but an object of G4ParticleDefinition is shared by all tracks of same type.

2.3.4 Step

Step has two points and also "delta" information of a particle (energy loss on the step, time-of-flight spent by the step, etc.). Each point knows the volume. In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume.

2.3.5 Trajectory

Trajectory is a record of a track history. It stores some information of all steps done by the track as objects of G4TrajectoryPoint class. It is advised not to store trajectories for secondary particles generated in a shower because of the memory consumption. The user can create his own trajectory class deriving from G4VTrajectory and G4VTrajectoryPoint base classes for storing any additional information useful to the simulation.

2.3.6 Physics processes

Geant4 has three basic types of physics processes as follows.

- "At rest process" (e.g. decay at rest) which is applied only for a particle at rest.
- "Continuous process" (e.g. ionization) which is accumulatively applied along a step of a particle.

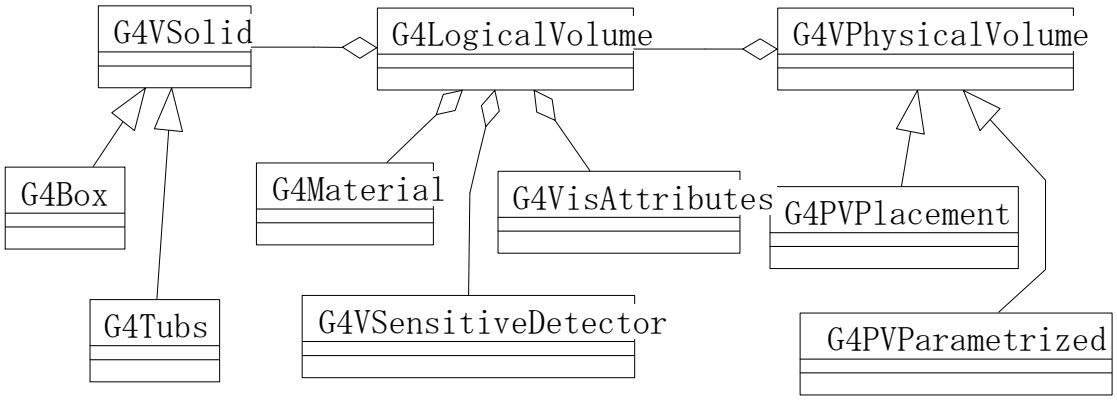

Figure 3. Classes which represent geometry

- "Discrete process" (e.g. decay on the fly) which is applied at the end point of a step.

Transportation is still a process which interacts with volume boundary. A process which requires the shortest physical interaction length limits the step.

In Geant4, the user defines cut-off by length instead of energy. It makes poor sense to use the energy cut-off. Range of 10 keV gamma in Silicon is around a few cm, while range of 10 keV electron in Silicon is a few micron. In Geant4, cut-off represents the range of secondaries. It does not mean that the track is killed at the corresponding energy. A track is traced down to zero kinetic energy except it meets to the other condition listed in 2.3.3. Additional "AtRest" process may occur even if the track becomes at rest. In case the energy corresponding to the given cut-off in a thin material is less than the available energy range of a physics process, Geant4 will not stop that particle by that process in the current volume (material). In case the track goes into another volume (material) which is more dense, that process may stop the track.

2.3.7 Geometry

Figure 3 illustrates three conceptual layers of classes for geometrical description.

- G4VSolid has shape and size. Various shapes and types of solid classes are derived from G4Vsolid base class.

- G4LogicalVolume has daughter volumes, material, sensitivity, user limits, etc.

- G4VPhysicalVolume has position and rotation. Objects of G4VPhysicalVolume can share an object of G4VPhysicalVolume if only their positions/rotations are different from each other.

2.3.8 Sensitive detector and hit

Each "Logical Volume" can have a pointer to a sensitive detector. Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector. A sensitive detector creates hit(s) using the information given in G4Step object. The user has to provide his/her own implementation of the detector response. Hit objects, which still are the user's class objects, are collected in a G4Event object at the end of an event.

2.4 Basic scenario how Geant4 runs

Geant4 has two major phases in its execution. The first phase is initialization, which is triggered by *Initialize()* method of G4RunManager, or an equivalence of UI command. Figure 4 is a scenario diagram of initialization phase. In this phase, construction of material and geometry according to the user's concrete implementation of G4VUserDetectorConstruction, and construction of particles,

physics processes and calculation of cross-section tables according to the user's concrete implementation of G4VUserPhysicsList occur.

The second phase is run, which is triggered by *BeamOn()* method of G4RunManager, or an equivalence of UI command. Figure 5 is a scenario diagram of the run phase. The run phase starts with closing and optimizing the geometry and them the event loop follows. Figure 6 shows the scenario flow inside the event loop.

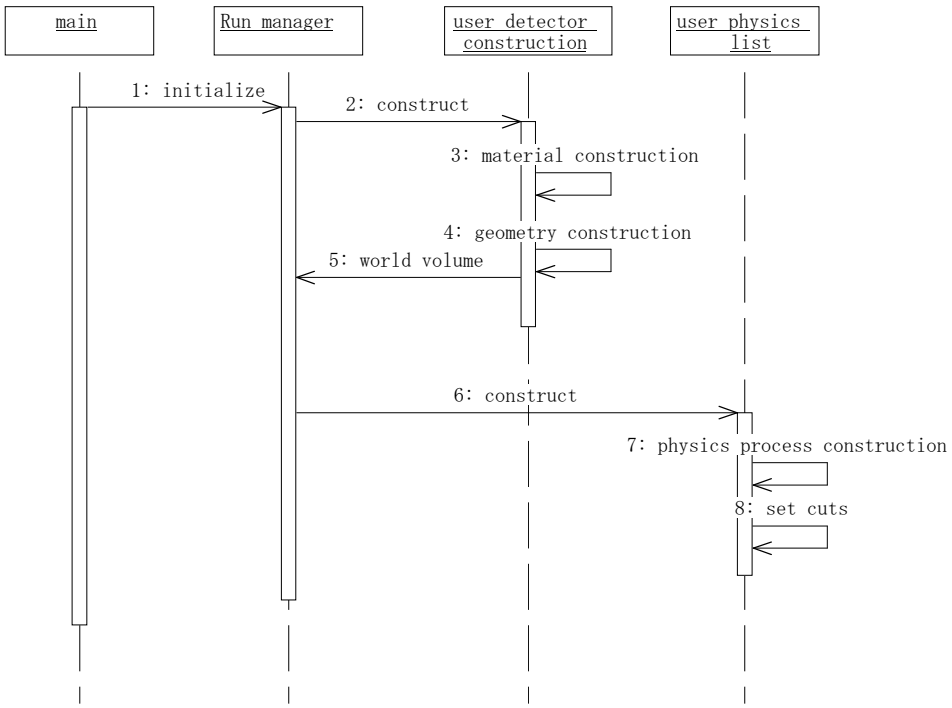

Figure 4. Scenario diagram for the initialization phase

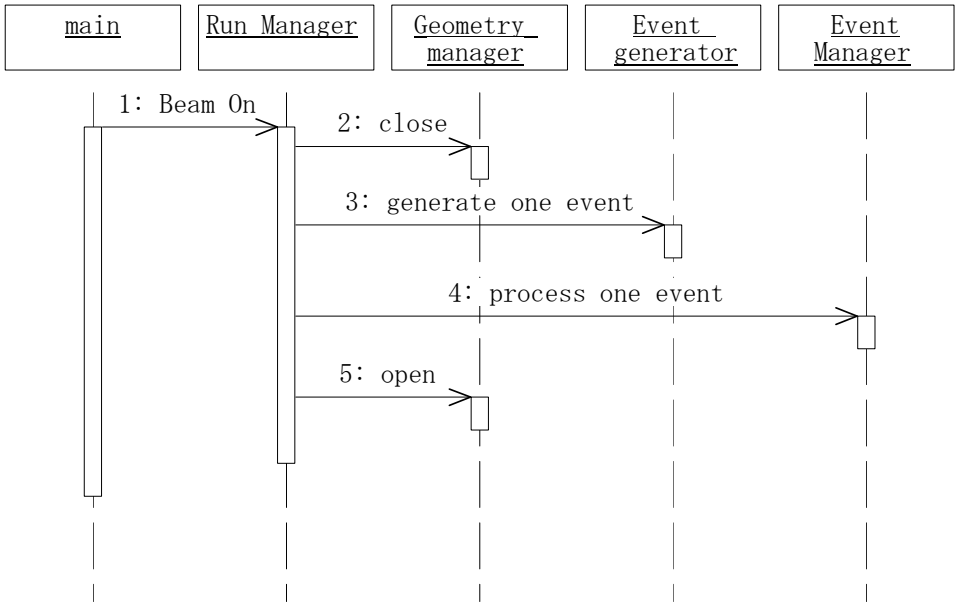

Figure 5. Scenario diagram for an event loop

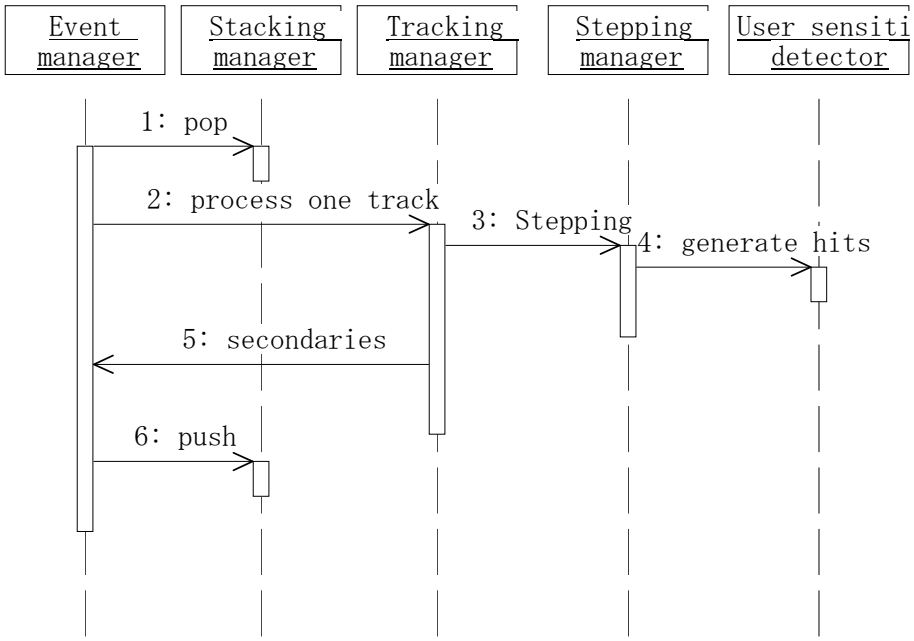

Figure 6. Scenario diagram for processing of one event

2.5 Unit system

Geant4 has no default unit. To give a number, unit must be “multiplied" to the number. For example :

double width = 12.5*m;

double density = 2.7*g/cm3;

Almost all commonly used units are available. The user can define new units. To get a number in a particular unit, divide a variable by the unit.

G4cout << dE / MeV << “ (MeV)” << G4endl;

3. THE MINIMAL THINGS TO USE GEANT4

In this chapter, one mandatory method and three mandatory classes that the user has to provide are explained. These mandatory classes must be derived from the abstract base classes provided by Geant4 and they must be set to G4RunManager.

3.1 *main()*

Geant4 does not provide the *main()* method. The user has to write his/her own *main()* method and, in it, the user has to construct G4RunManager or its derived class, and has to set three mandatory classes inherit G4VUserDetectorConstruction, G4VUserPhysicsList and G4VUserPrimaryGeneratorAction. The user can define Visualization Manager, (G)UI session, optional user action classes, and/or persistency manager in his/her *main()* to customize the behavior of application based on Geant4.

3.2 Detector construction

The user has to derive his/her own concrete class from G4VUserDetectorConstruction abstract base class. In the virtual method Construct(), the user has to

- Construct all necessary materials
- Construct volumes of his/her detector geometry
- Construct his/her sensitive detector classes and set them to the detector volumes

Optionally the user can define visualization attributes and/or step limits of his/her detector elements.

3.3 Physics list

Geant4 does not have any default particles or processes. Even for the particle transportation, the user has to define it explicitly. To define particles and processes, the user has to derive his/her own concrete class from G4VUserPhysicsList abstract base class. In the concrete implementation, definition of all necessary particles and all necessary processes must be declared and proper processes must be assigned to proper particles. Geant4 provides lots of utility classes/methods to make the user be easy to define. Also in this concrete implementation, cut-off values must be declared.

3.4 Definition of primary generators

The user has to derive his/her concrete class from G4VUserPrimaryGeneratorAction abstract base class and to pass a G4Event object to one or more primary generator concrete class objects which generate primary vertices and primary particles. Geant4 provides two generators, those are G4ParticleGun and G4HEPEvtInterface, which interfaces to /hepevt/ common block via ascii file. An interface directly to C++ version of PYTHIA direct interface will be available quite soon. Also, an interface to HepMC is planned.

4. OPTIONAL USER ACTION CLASSES

Geant4 provides five user's optional "action classes" to control the behavior of Geant4. In this chapter, these classes are explained with probable usages. All user action classes, methods of which are invoked during "Beam On", must be constructed in the user's *main*() and must be set to the RunManager.

4.1 G4UserRunAction

This base class defines two methods, *BeginOfRunAction(const G4Run*)* and *EndOfRunAction(const G4Run*)*. These are invoked at the beginning and the end of each run. Most probable use-case of these methods are booking and manipulating histograms.

4.2 G4UserEventAction

This base class defines two methods. The first method is *BeginOfEventAction(const G4Event*)* and it is invoked at the beginning of each event processing but after the generation of the primary particles. Thus the G4Event object already has primary particles and they can be used for the event filtering. The second method is *EndOfEventAction(const G4Event*)*, which is invoked at the end of each event. Basic analysis of the event can be implemented in this method.

4.3 G4UserStackingAction

This base class defines three methods. The user can customize the stacking mechanism by implementing his/her own concrete class derived from this base class. In Geant4, three are three stacks, which are called as "Urgent stack", "Waiting stack" and "Postpone to next event stack", respectively. When a new track is pushed to the G4StackManager, regardless of it is primary or secondary, the method *ClassifyNewTrack(const G4Track*)* is invoked. The user can classify the track as one of the following four classifications.

Urgent : the track to be sent to the "Urgent stack"

Waiting : the track to be sent to the "Waiting stack"

PostponeToNextEvent : the track to be sent to "Postpone to next event stack"

Kill : the track to be killed immediately without pushing to a stack

The track is always popped from "Urgent stack" and sent to G4EventManager. Once this stack becomes empty, the method *NewStage()* is invoked. The user can re-classify the tracks stacked in the "Waiting stack".

4.4 G4UserTrackingAction

This base class defines two methods, *PreUserTrackingAction(const G4Track*)* and *PostUserTracking Action(const G4Track*)*, which are invoked at the beginning and the end of each track processing, respectively.

4.5 G4UserSteppingAction

This base class defines one method of *UserSteppingAction(const G4Step*)* which is invoked at the end of procedure of each step. The user can kill the track, suspend the stepping and send it back temporary to the stack, or suspend the stepping and postpone it to the next event.

5. USER SUPPORTS

5.1 Documents

Geant4 provides various documents. All of them are accessible from the Geant4 official Web page[1]. The documents and examples are continuously evolving. Currently, the site has six documents as follows.

- Introduction to Geant4 : This document
- Installation guide : This document explains how to install Geant4 toolkit onto various types of machines, at the same time required external libraries and the way of getting them are summarized.
- User's guide for application developer : This documents explains the basic usage of Geant4 toolkit suitable for the user who is developing a simulation program based on Geant4.
- User's guide for toolkit developer : This documents explains how to add/alternate some built-in functionalities of Geant4, such as adding a new physics process, adding a new shape of solid, etc.
- Physics reference manual : This document explains in deep what kinds of physics processes are implemented in Geant4 toolkit and what are the referenced materials.
- Software reference manual : This document lists all of Geant4 class definitions. This document is available only on Web and required page is generated automatically by access.

5.2 Examples

Geant4 provides various examples. All of them are kept updated every release and continuously evolving. Especially, examples for new areas of applications rather than high-energy physics are glowing according to the number of the users and collaborators in these areas. There are three categories of examples.

- Novice examples : These examples are aimed to learn how to use Geant4 toolkit.
- Extended examples : These examples demonstrate how to use Geant4 toolkit with some other external packages such as histogramming, persistency, etc.
- Advanced examples : Each example in this category demonstrates how to use one or two category in Geant4 in detail. Some examples in this category are converted from our global system tests.

5.3 Other user supports

In our Web site, the bug reporting system is provided. Once a user encounters to a bug, he/she can report it to this system. The coordinator of corresponding category will investigate the report and the

reporter will get feedback within a few days. Web-based source code browser and the user's news group will be in public quite soon.

6. SUMMARY

Geant4 is one of the first successful attempt to re-design a major package of HEP software for the next generation of experiments using an Object-Oriented environment. A variety of requirements also came from heavy ion physics, CP violation physics, cosmic ray physics, medical applications and space science applications. Geant4 is the first software development in HEP which fully applies most recent software engineering methodologies. Because of adoption of Object-orientation, wide coverage of physics processes, geometrical implementations, etc. is realized. At the same moment, user's extension to area is easily achieved. Geant4 provides various user supports and they are continuously evolving. Geant4 is not only for HEP but goes well beyond that. Intensive use of Geant4 in various fields has already been started.

ACKNOWLEDGEMENTS

The author wishes to thank the organizers of the school for their supports on this lecture. The author wishes to thank Dr. Gabriele Cosmo (CERN/IT) and Dr. Marc Verderi (LPNHE) their most useful comments on this lecture and also their assists on the associated exercises. The author acknowledges all of collaborators of Geant4 for their individual excellent works. All the members of the collaboration who should share the authorship of this lecture are listed on our Web site.

REFERENCES

[1] Geant4 Home page. http://cern.ch/geant4/

[2] S. Giani, et. al. "Geant4: An Object-Oriented toolkit for Simulation in HEP", CERN/LHCC 98-44.

[3] K. Amako, et. al. Users requirements document. The latest version can be found from the Geant4 Web site.