GeantV interfaces

A. GHEATA

GEANTV WEEKLY AUG 8, 2017

Input configuration: Messengers?

Input configuration is an important user interface for simulation

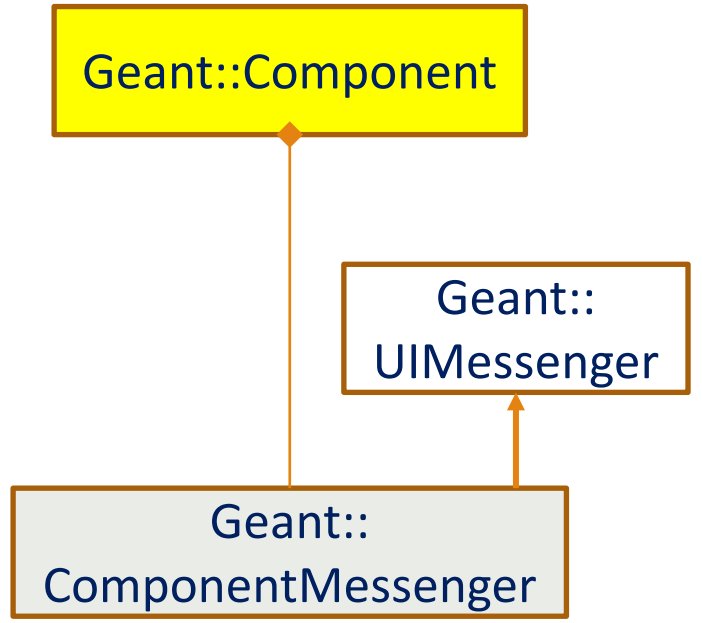
- Allowing user to set parameters related to any configurable component/category:
  - Run, event generator, detector construction, visualization, …

Geant4 implementation: UI messengers per component, registered to UI manager

- User starts UI, then issues “commands”, which can form a macro

Evolution in GeantV:

- version 2: hard-coded parameters in the executable
- current: the same, but providing a steering script where parameters can be configured
- To do: messengers, something else?
  - A native configuration mechanism needed besides the possibility to configure GeantV using an external package (e.g. DD4HEP)




```
/testem/setAbsMat G4_Au
/testem/det/setAbsThick 9.658 um
```

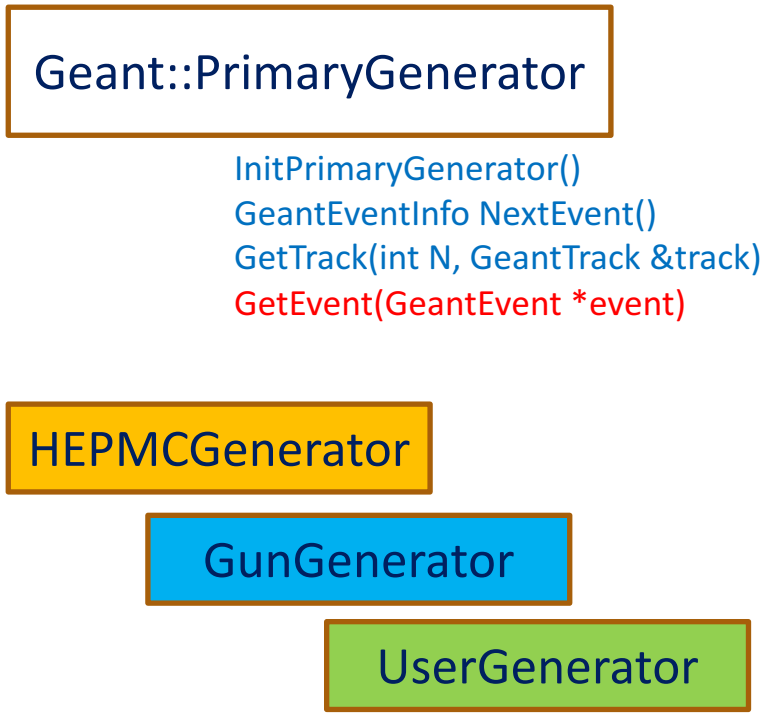
Input: User generator

User-defined generator: adding events one-by-one to GeantV

- Functionality equivalent to G4VPrimaryGenerator
- Extra concept in GeantV: event slot
  - no more than nslots events transported concurrently

Implementation

- version 2: called concurrently after freeing an event slot
- version 3: called by main thread at initialization and filling event server (concurrent service)
  - memory problems for many input events
- To do:
  - maintain nslots buffer + queue of pending events (fixed max size); activate queued event in the server once a slot is released
  - support external event loop where events are inserted externally, without invoking a user generator (needed for CMSSW)



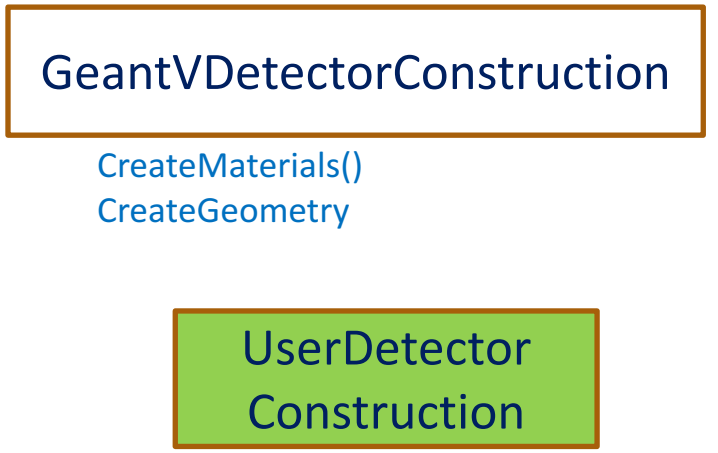
Input: Geometry

Geometry definition interface: detector construction

- Quite similar to Geant4 functionality
- Purpose: whatever the input, end-up with VecGeom transient geometry structure, connect basketizers to volumes, connect GeantV regions to logical volumes

Evolution:

- version 2: no detector construction class, geometry loaded from ROOT file (or GDML->ROOT), then converted to VecGeom, no regions
- current version: detector construction still supporting geometry construction via ROOT transient geometry, supporting Regions. Material conversion from ROOT to GeantV materials included as lambda in the detector construction
- Supporting geometry definition via external package (e.g. DD4HEP) possible by emulating native detector construction (as for Geant4)
- To do's: ?



Input: physics configuration

Based on physics list, same as in Geant4

Configuration parameters now hardcoded (or can be passed as macro arguments)

- Same as general configuration issue

Mihaly: investigating the possibility to work with multiple physics lists, not for changing the physics per region, but e.g. sampling tables vs. rejection, fast versus detailed algorithms

To do: ?

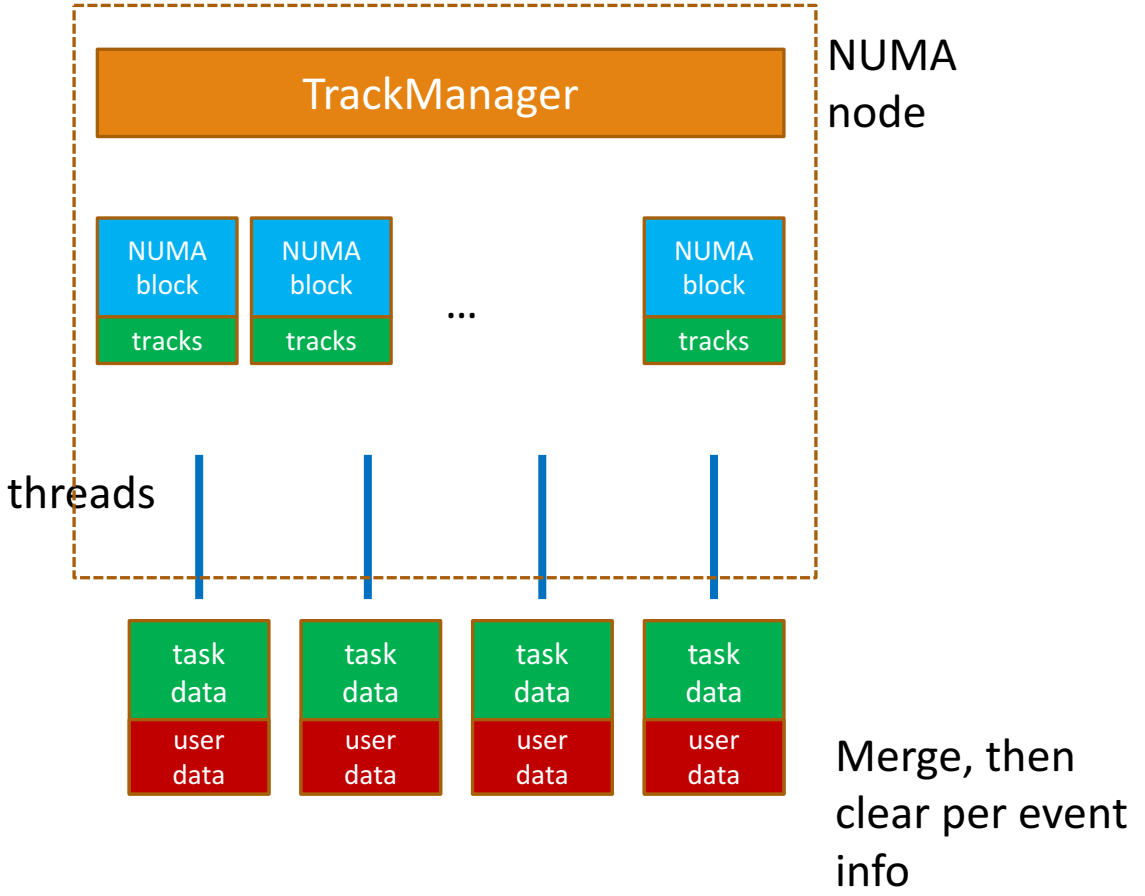
User data registration in GeantV

version 2: User data per step (“snapshot” hits)

- e.g. position/momentum for some particles in a given detector
- generating possibly large amount of data, but generally used only for debugging purpose
- Mechanism using ROOT I/O and parallel merging provided
  - Still not disentangling events

version 3: “Summable” hit information

- e.g. total energy deposit per event in a calorimeter cell
  - Custom user data organized per event slot has to be registered by the user application, then attached to task data
  - Filled during SteppingActions, the information gets merged automatically per event, then cleared
  - Inclusive information (per run, e.g. total number of tracks of a given type) has now to be summed-up manually in thread-safe manner



User data management

MyApplication::Initialize()

- Register summable user data types (implementing functions Clear() and Merge())
  - TaskDataHandle<UserData> *handle = fRunMgr->GetTDManager()->RegisterUserData<UserData>(const char *name)
  - Keep handles as data members of the user application

MyApplication::AttachUserData(GeantTaskData *td)

- Called by every task/thread in the initialization phase, must create UserData objects (as many as needed, e.g. per detector) and attach to task data objects (per event or per run) via the handlers
  - fDataHandleEvents->AttachUserData(new UserDataPerEvent, td);
  - fDataHandleRun->AttachUserData(new UserDataPerRun, td)

MyApplication::SteppingActions(GeantTrack*, GeantTaskData *td)

- Retrieve user data per event slot from task data, then score per event slot information
  - UserDataPerEvent *myData = (*fDataHandleEvents)(td)->GetDataPerEvent(track->fEvslot);

MyApplication::FinishEvent(GeantEvent *event)

- Called by a single worker/task; merge user data per event then clear it:
  - fRunManager->GetTDManager()->MergeUserData(event->GetSlot(), *fDataHandlerEvents);
- Sum-up per run info in a thread safe manner

MyApplication::FinishRun()

- Analyze collected data

Track data management

Physics processes have state data dependent on step, to be attached to tracks

- Tracks are allocated contiguously in blocks

Register user-defined class to TrackDataMgr singleton

- TrackToken *token = TrackDataMgr::Instance()->RegisterDataType<ModelData>(“someName”);
- In-place construction is recorded as lambda

Access data run time

- ModelData * myData = token->Data<ModelData>(track);

Any number of the same data type can be added

- When clearing a track run-time, the in-place constructors are avoided, we rather call copy from a cleared blueprint track

The mechanism can be used for any model needing it

Run, primary & stepping interfaces

Very similar to Geant4

- Begin/EndRun(), called by main thread
- Begin/EndEvent(), called by a single worker thread (thread safe for event data)
- Begin/EndPrimary()
- SteppingActions(), coming with scalar and vector signatures

The full state can be queried from the track. Currently no pre/post step information, so user cannot query e.g. momentum of track before step

- To be added

Discussion

Missing features? Things to clean-up.

Changed due to task-based approach?

- Hopefully not

Interfaces for handling MC information – not yet discussed