Geant4 Geometry Objects Persistency using ROOT

Witek Pokorski

13.07.2005

Outline

- Underlying idea
- Overall approach
- Technical issues
- Practical case
- Conclusion

Underlying idea

- essential part of any Geant4 simulation application: geometry description
  - probably constitutes the main part of 'data' which needs to be loaded
  - in case of complex geometries can take essential part of the initialization time
- Geant4 does not come with any persistency mechanism for the geometry objects
  - the Geant4 geometry tree has to be 'rebuilt' each time
  - Geant4 geometries are often created by converting from experiment-specific models which makes then 'non-exportable'
- our goal: provide way of quick saving and reading back the G4 geometry in/from a (binary) file
  - would nicely extend the functionality of the toolkit

Some remark...

- this is a different use-case from GDML, where universality of the format was top-priority
  - GDML allows interchanging geometries between different models (Geant4, ROOT, etc)
    - it's 'flavor-free', no application-specific binding
  - GDML can also be used for implementing geometry
- ROOT persistency for Geant4 allows saving Geant4 geometry (G4 objects) and reading it back into a Geant4 application
  - one could still use it to interchange geometries between different Geant4 applications
    - save LHCb geometry in Gauss and then load it into any other G4 application to run tests of visualize
    - it would make Geant4 applications less 'geometry-bound', extend the spectrum of their usage

Overall approach

- use **lcgdict** tool to create Reflex dictionary for the Geant4 classes
  - fully non-intrusive
  - can by fully automated (all done in Makefile)
  - requires only selection.xml file with list of classes
- **Cintex** tool allows to convert Reflex dictionary information into CINT data structure
  - Cintex will not be needed once CINT is able to interact directly with Reflex dictionaries
- use ROOT I/O to save the geometry tree into .root file
  - create a simple wrapper class containing pointer to top volume
  - call WriteObject method for that object
  - ROOT I/O saves all the geometry tree by following the pointers

Technical Issues (1/3)

- ROOT I/O requires all the (persistent) classes to have default constructors
  - they are used to allocate memory when reading back the objects
  - they need to initialize all the pointers (non-null pointers are considered by ROOT I/O as valid ones and are not overwritten)
  - most of the Geant4 geometry classes do not have default constructors...
  - default constructors can be added to Geant4, but should never be called from the users' code
    - constructors in Geant4 perform different kinds of registration which are not possible in the default case

Technical Issues (2/3)

- variable length arrays of objects are not (yet) supported by ROOT I/O
  - Geant4 uses quite often arrays of non-fundamental types
    - only solution for the moment: move to std::vector
- variable length arrays of fundamental types are supported but header files need to be instrumented

  double* x; //[N]

  needs to be added in the class definition
  - could be moved to the selection files (lcgdict) in the future

Technical Issues (3/3)

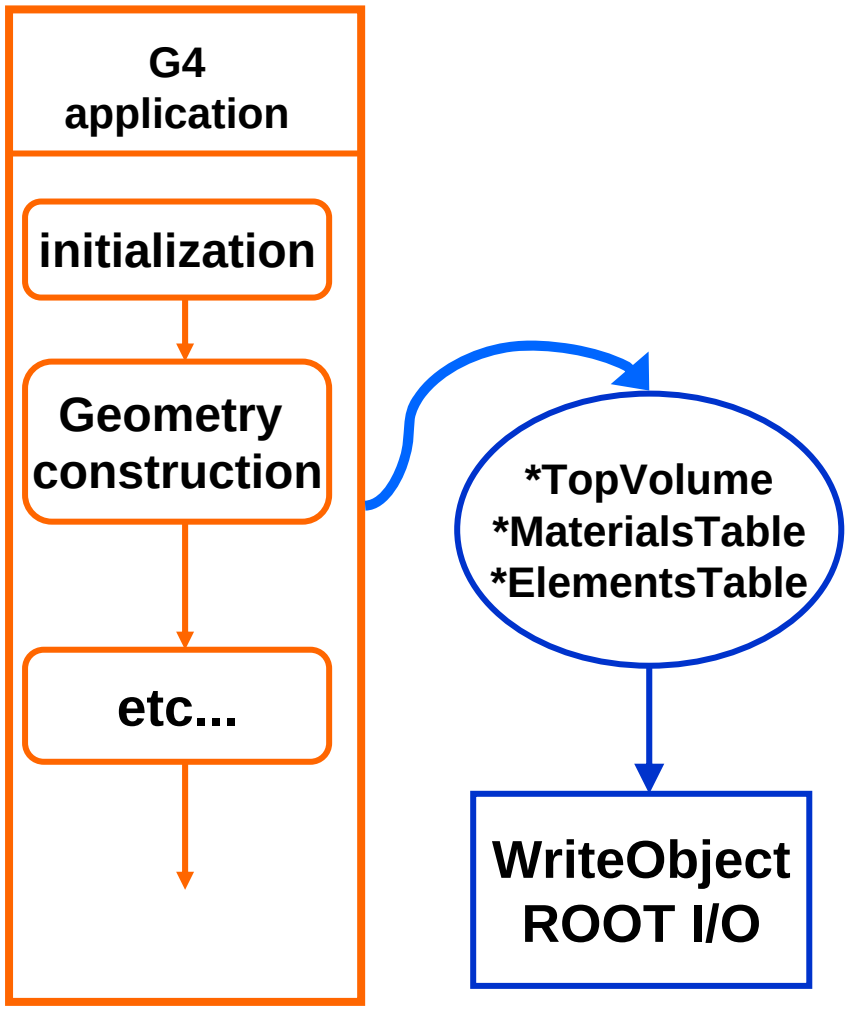
- a few specific issues
  - **struct** with members being pointers should have default constructor initializing the pointers
  - typedef struct {...} MyStruct;

  should be replaced by

  struct MyStruct {...};
    - lcgdict fails to produce a sensible name for the destructor for anonymous struct
  - MyClass** should be replaced by std::vector<MyClass*>
    - ** is ambiguous from the point of view of persistency; it can be a pointer to an array or an array of pointers

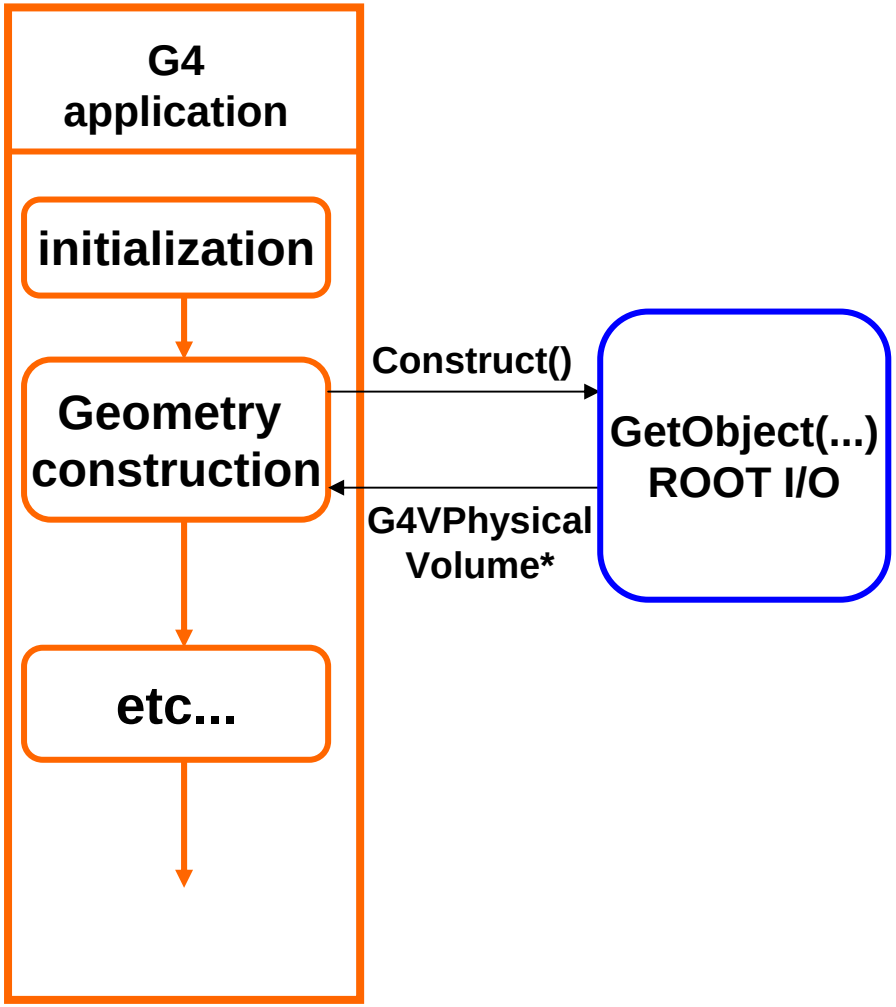
Practical case - writing

- we have our Geant4 geometry (say LHCb) in memory and we want to save it in .root file
- we call a simple 'GeoWriter' tool which:
  - creates a 'wrapper' object containing *TopVolume and pointers to materials and elements static tables
  - calls WriteObject ROOT I/O method
- trivial implementation
  - no any 'scanning' of the geometry tree needed
  - ROOT traverses all the geometry tree and stores it
  - only needed thing is to export the pointer to the top volume



Practical case - reading

- only binding to ROOT in DetectorConstruction class
- the Geant4 'main' does not see the loading of the geometry using ROOT I/O
  - 'standard' DetectorConstruction replaced by ROOTDetConstr.
  - G4VUserDetectorConstruction ::Construct() returns pointer to the top volume
- ROOTDetectorConstruction as a simple 'plug-in'
  - one just needs to instantiate it from the 'main'



Remark on Python interfacing

- once the dictionary for Geant4 classes is there, Python binding comes for free
  - PyLCGDict/PyReflex/PyROOT allows to interact with any Geant4 as well as ROOT class from Python
  - Python ideal to glue different 'worlds' (Geant4, ROOT, GDML, etc) together
  - see http://lcgapp.cern.ch/project/simu/framework/PYGEANT4/pyg4.html for simple examples
- saving/loading G4 geometry using ROOT I/O even trivial from Python prompt
  - modularization of 'reader/writer' more natural
    - less explicit binding

Final remarks

- simplest case discussed here - only geometry (and materials) stored in ROOT file
  - next steps (if needed) could include saving optimization information (voxels?), material cuts tables, etc
- necessary changes in G4 implemented (on my local disk....) and tested for LHCb
  - entire LHCb geometry & materials file ~220kB
  - writing and reading back time - less than 2 seconds

- http://lcgapp.cern.ch/project/simu/framework/G4ROOT/g4root.html (working document)

Conclusions

- dictionary for Geant4 classes is an essential element both from the point of view of persistency as well as interactivity
  - proposal: generate (and store on AFS) Reflex dictionary for every new Geant4 release
- geometry objects persistency using ROOT comes (almost) for free
  - would be a very nice additional functionality for the toolkit
  - proposal: release LCG-internal version of Geant4 with all the necessary changes (and the dictionary) to allow users to give it a try
- once the dictionary is there, Python interfacing comes (fully) for free with PyLCGDict/PyReflex/PyROOT
  - another argument in favor of releasing the dictionary
  - Python environment perfect for Plug&Play