Atlas & Geant4

The Development of the ATLAS Simulation Framework

The Atlas Detector and its Full Simulation

Adele.Rimoldi University of Pavia & INFN, Italy
adele.rimoldi@cern.ch
Introduction

- Ten years of "requirements", gathered running the FORTRAN-based simulation (Geant3)
- The simulation package for an LHC experiment is a very complex system
  - Detailed simulation
  - Testbeams
  - Physics studies
  - Fast/semi-fast simulation
Design principles

- **Scalability**
  - The simulation package must be able to accept:
    - the simplest testbeam simulation
    - the complete detector simulation

- **Clear separation**
  - between "core" software and user implementation
    - the "framework"
      - does not depend on what the users want to do
      - provides maximum flexibility and freedom to the user who wants to implement their own code
      - must be lightweight

- Implement some middleware on top of G4 for extending its functionality
- Dynamic loading + plug-in techniques for connecting user’s code
- Action on demand to avoid wasting memory
The simulation program

- has no geometry
  - Users build it interactively at initialization time

- has no physics
  - Users choose from a catalog, depending on what they want to do
  - ..or implement their own, and add it to the catalog

- has no kinematics
  - See above...

- is an empty box

The best framework is no framework at all
Steps

- First of all, dismantle the G4 framework
  - no UserDetectorConstruction, no UserPhysicsList…

- Provide a set of services that users can refer to
  - MaterialManager
  - DetectorFacility
  - PhysicsListCatalog
  - VisCenter…

- Use the Geant4 (G)UI facilities for re-creating all connections

- Think in terms of **dynamic loading**, all the time

- **Save memory**, as much as one can, use proxies and lightweight objects (memory has a price, after all).

  *a slim application is the final goal*
Example - the Material Manager

- Materials must be defined centrally (in a DB, or in an ASCII file). People must use the same materials throughout the Simulation Program.

- A **MaterialManager class** provides all services for getting all materials in the simulation and for permitting users to access them.
  - Users don’t redefine materials all the time but they *access them* via the MaterialManager.

```c++
MaterialManager *mm=MaterialManager::GetMaterialManager();
G4Material *Fe=mm->GetMaterial("Iron");
```
Example - the material manager

- A table containing hundreds of G4Materials has a (big) cost (both in terms of memory and time spent initializing the program), especially if you then don’t use them

  ◆ **Lazy instanciation**
  
  ★ create a G4Material only when you need it for creating a volume but keep a lightweight representation of all material you have in your DB, for permitting the user to choose

  ◆ **Reference Counting**
  
  ★ keep track of how many volumes point to the same G4Material, destroy the G4Material as soon as no volume points to it anymore

- This requires wrapping the G4Material with some specific classes of ours (**AtlMaterial**, **AtlMixture**...)}
Geometry organization

- Geometry is naturally split into *Detectors*
  - A *DetectorFacility class* is provided for this
  - Users must inherit from DF and (in principle) just implement a `BuildGeometry()` method

- There is no predefined detector organization, users build their experiment at run time
  - Atlas logically contains the Atlas IDET, but you can make it such that the IDET contains Atlas

- Users register their detectors into a catalog by just adding one line of code

```cpp
static DetectorFacilityEntry<TileSection> calo("TileSection");
```
Geometry Organization

- DetectorFacilities are accessible at run time through the catalog
  
- and define their own UI directory
  
- and can be combined together in the way you want
  
- and then moved around

- OK, *eventually* you have to decide who’s the boss
Geometry Organization

- **Advantages**
  - *There is no predefined geometry*
  - *Different representations of the same detector* can co-exist in the same program
  - Users can migrate their detectors to a general purpose simulation program without breaking it
  - The amount of *memory* taken by the geometry is reduced to a minimum
  - Dynamic loading can be used for bringing detector geometries in

- **Disadvantages**
  - *There is no predefined geometry*
  - You must be *careful when assembling a complicated detector*
  - there is *some typing to do* (but you can save everything in a macro)
The Geometry wrapper

- A set of classes (AtlLogicalVolume, AtlPhysicalVolume, AtlRotationMatrix etc..) wrap up the corresponding G4 classes, extend their functionality and self-register on creation to a GeometryManager which keeps a handle to all geometry objects as organized by detector.

- At this point one can:
  - change the visualization attributes of a volume interactively
  - change positions and rotations interactively
  - change physics cuts for a certain volume
  - assign sensitive detector objects to volumes interactively
  - remove complete detectors/detector trees
    - by removing the plug-in, for instance
Sensitive detectors

- Abstract interface
- They are organized as plug-ins, that an user attaches when they are needed
- Assigned interactively to the volumes they must act upon
  - A volume can become sensitive at run time
  - General purpose SDs can be implemented for specialized actions
    - to calculate the thickness in R.L. of a sub-detector
Physics

- Geant4 physics lists are great for physics customization but generate quite some confusion in the “normal” user
- hence:
  - Physics list are proxied
  - a Physics list catalog provides a list of the most common lists to choose from (EM, EM+Had…)
  - New physics lists can always be added (dynamic loading + plug-in)
- Normal users choose the most appropriate
- Advanced users can define their own...
Event Generators

- Abstract interface based on the HepMC format
  Generator $\Rightarrow$ HepMC $\Rightarrow$ Geant4
- Plug-ins
- Abstract interface for **editing/filtering particles** and **vertices** (plug-ins)
  - Vertex displacement
  - $\eta/\phi$ filtering
  - energy/particle type filtering
- Primary event (signal)
- Secondary events *from a different generator*
- Pile-up at the generator level
User Actions

- Abstract interface which combines all G4 user actions (and more) into a single class that users must inherit from.
- Plug-ins
- Actions must now be registered interactively
- Several UAs can now be run concurrently
  - Priority mechanism in place to decide which comes first (if needed)
Toward a software environment?

- Users are not supposed to build their application anymore, they just run an existing one (à la PAW/ROOT)
  - No need of complicated tools for cvs check-out, building libraries, creating executables
  - Highly configurable
  - Maximize code “re-use”, sharing etc..
  - Some automatic code generation possible..

- On the other hand, though
  - proliferation of libraries, add-ons, plug-ins...
  - need for an accurate versioning system
  - makes for an interesting life for coordinators...
To make a point

- ~100 classes, developed on Linux, being ported to Solaris, clients knocking at the door
- It is there, it works…
  - continuous development
  - lack of documentation
  - code to be cleaned up
- Nothing is Atlas-specific
- if Geant4 runs, Goofy will run
  - the only additional package needed is HepMC
- Now concentrating on the implementation side
The Atlas Detector

- Four main subdetectors
  - **Inner detector** - momentum measurement of charged particles
    - high precision silicon trackers
    - straw tracker with TR capability
  - **Calorimeters** - energies carried by particles measurements
    - Lar EM and Endcap calorimeters
    - Scintillating Tile calorimeter
  - **Muon spectrometer** - muon identification and measurement
    - High precision Drift Tubes for tracking, RPCs and TGCs for triggering
  - **Magnet system** - bending of charged particles for momentum measurements
An Example of Implementation: The Muon Spectrometer

- Geometrical information
  - Read by an external file
  - Handled by a set of classes that store position, size, composition
- MDT, RPC, TGC systems all simulated in full details
  - Magnet implementation undergoing
- Currently in production
- Already interfaced to the muon reconstruction program

A/H -> $\mu\mu$ @150 GeV
EM Barrel Calorimeter

- A 50 GeV $e^-$ in the Atlas EM barrel calorimeter
The Inner Detector

- $H \rightarrow 4 \mu$ event

Progressive implementation
- Simplified detectors $\Rightarrow$ realistic geometry
- Sensitive detector implementation + digitization
- Check against existing G3 simulation
Event Generators

- Different event generators implemented, to be used for different purposes
  - Single particle generator ($4\pi$)
  - Test beam
  - Grid generator (for scanning uniformly)
  - Pythia (vs. 5 and vs. 6)
  - From ASCII
- Particle filters
Frame Facilities and Analysis

- Abstract interface to analysis systems for implementing some histogramming capabilities
- Currently implemented
  - *Hbook*
  - *Root*
  - *HTL*
Conclusions & Plans

- Implementation of the
  - different ATLAS Subdetectors undergoing
  - as well as testbeam setups
    - As test case for a detailed and realistic geometry
- Magnet system under implementation
- Event Generators in place
  - background to be added
- Extensive event production (@CERN and outside)
  - already successfully completed (10**7 ev) (phase 0)
- Easy and successful installation and running
  - Simple instructions on the web pages