The GEANT4 Experience — Panel Discussion on The Transition to OO — CHEP97

John Allison

University of Manchester, UK

1 The GEANT4 Experience

Most of us in the GEANT4 team came with little experience in object oriented methods (OOM). On the other hand, many had been involved in various aspects of particle physics computing for years and thus brought a wealth of general experience and skills; quite a number had been involved in GEANT3. And we passionately believed that if GEANT was going to be useful in the LHC era, it had to be completely re-written. GEANT3 had become unmaintainable or, at least, undevelopable. And we believed the OO techniques were required.

Much of the first year of the project was devoted to design. We adopted the Booch OO design and analysis concepts and notation, with the help of Rational Rose design tool but, from the first, it was recognised that it was also necessary to write prototype code in order to evolve the design (the philosophy of iterative evolution) and an early decision was made to use C++.

This turned out to be typical of the common sense decisions and philosophy simply "to make a start" and "to learn on the job", with help from attending courses, reading and simply using the knowledge of those with more experience. A requirements document was written (and published and publicised) — and the exercise helped clarify our objectives — but it contained little that was not "obvious" to most particle physicists, especially any with experience in GEANT3. I doubt if many read it, once it had been written.

Thus began a very pragmatic distributed development process, with little formal analysis but with much trial and error (iterative evolution), many email exchanges, frequent meetings and a great deal of common sense, intelligence and good will. It has been an enjoyable experience. We have reached the "alpha

Preprint submitted to Elsevier Preprint *13 May 1997*

tag" (a limited release) in about 2 years, roughly comparable to GEANT3 in functionality, and incorporating clever new geometry search algorithms, many improved physics processes and a more general tracking algorithm. This is quite an achievement. We have given ourselves another 2 years to produce a usable product.

What else do I want to say about GEANT4 development process?

- The project was divided into "categories" or "domains" — digits+hits, events+tracks, g3tog4, geometry, global, graphics_reps, intercoms, interfaces, particle+matter, tracking, visualization — in total 1248 files and 209211 lines (a snapshot just before the alpha tag).
- Simple Gnu makefiles were made for each domain and sub-domain, each including GNUmakefile.architecture and GNUmakefile.common. This has served its purpose but a more sophisticated build and release algorithm is needed.
- The code is maintained by CVS. CVS is more than a code repository; it is a tool for parallel code development, offering utilities for merging and versioning. It is indispensable.
- We have used and made extensive contributions to CLHEP, the C++ class library for particle physics.
- We use Rogue Wave tools.h++ template container classes (but will probably migrate to STL); such classes offer huge productivity gains and are very efficient.

The early releases were extremely immature, but informative enough for major experiments, such as BaBar, to plan for adoption. We hope that the alpha tag will be good enough to help people start coding their detector model and seeing some results (but, of course, it is still "at your risk").

Here are some thoughts on the product:

- GEANT4 is highly object oriented
- GEANT4 contains some highly sophisticated OO designs, particularly in geometry, tracking and intercoms
- GEANT4 contains a lot of physics, encapsulated in particle and process classes
- The abstract visualization interface seems to work
- The OO Method is essential
- Categories developed more or less independently work together without serious conflicts/clashes/adverse interactions
- Abstract interfaces work
- Inheritance and code re-use works
- OOP leads to high number of small code units (functions)
- OOP leads to large number of files

- CVS essential
- OO Design tools have been used successfully
- Performance is good

...and about people...

- ...thoughts about the GEANT4 team: in spite of initial inexperience on the part of most members (only one or two had previously been taught OO)...
  - All were experienced and established in computing
  - All were/are committed to OO
  - All were willing to put effort into learning OO
  - Most learnt "on the job"
  - All are now adequately proficient in C++ OO
- ...and thoughts about physicists in general
  - Physicists *can* learn OO Methods
  - There has to be a way of *easing* people into it.
  - Physicists are good at learning on the job
  - Needs dedication to learn full OO
  - Can use "recipe" methods to begin with
  - Start with "user defined types"
  - Needs 2 years to grasp concepts of OO and adequate mastery of language!
  - Need to acquire new way of thinking
  - It just comes with practice — difficult to teach
  - It comes more naturally to some than others
  - Once acquired, one feels it is the "natural" way of thinking
  - Young people will know nothing else
  - It's fun

My conclusion? Using object oriented methods is definitely the way to go!