Fabiola Gianotti

PHYSICS and SOFTWARE

- Software requirements of physics groups

- What should

  Detector Performance
  Combined Performance
  Physics
  } groups

  do for software

Note : first thoughts ….
comments/suggestions/criticisms welcome

Physics requirements

Each issue has its specific requirements
→ not discussed here. However:

Three main/general requirements:

❶ Use past experience:
~ 10 years of performance/physics studies
→ deep knowledge of detector performance, reconstruction algorithms, physics analyses (documented in various TDR …).
This should not be lost, but should be transferred to the new software.

Ex. : "Code from scratch in full C++/OO'' with no reference to this previous experience does not satisfy this requirement.
"Reverse engineering of existing Fortran code as first step to C++" does satisfy this requirement

❷ Performance:

new software should provide expected detector/physics performance.
Performance/physics evaluation is the first serious benchmark.
Reference: Physics TDR

❸ Simplicity /functionality:

new software must be "as simple and functional" as possible:

-- aim is physics and not software development "per se"
-- "end-users" are not "blind" users of a black box but developers → should have easy access to most of software
-- each member of ATLAS (and not a few elected people) should be able to do analysis at LHC (w/o help of a software engineer)

In turn everybody should be ready to learn and improve his/her way of producing software

---

Contributions of physics/performance groups to software:

- MC generators
- event simulation (fast, full, intermediate ?)
- reconstruction
- graphics/event display
- analysis tools
- others …

MONTE CARLO GENERATORS

- New Physics group convened by I. Hinchliffe
- Will have many MC generators in ATLAS
- Activity related to software:
  - -- fit to overall architecture
  - -- transparent use of all generators
  - -- allow multi-language:
    Fortran (ISAJET)
    C++ (Pythia, Herwig being rewritten)
  - -- give inputs to authors:
    - -- same classes for all C++/OO generators ?
    - -- define common output structure in same spirit as HEPEVT common
    - -- as uniform datacards as possible … ?

SIMULATION

Today two lines:

- full simulation (GEANT): detailed but CPU consuming
- fast simulation (ATLFAST): fast but very simple (particle smearing, no shower shapes)

Future: need in addition intermediate step between ATLFAST and GEANT

- could be an improved ATLFAST including shower parametrisations (done in part by K.Jakobs et al. for trigger TP)
- could be a simplified GEANT including shower parametrisations

Shower parametrisations:

→ understand for which particles, over which η /E range showers are parametrised.

→ lot of work for Detector/Combined Performance/ Simulation groups (test beam data, other experiments)

→ implications for software

Notes:

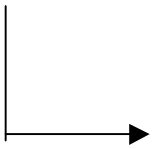
1) Example of further physics requirement: must be able to overlap simulated events with real data (e.g. for processes where MC generators or detector simulation are not adequate)

2) COB discussion:
   -- make use of ATLFAST++ without further development
   -- in parallel, a new version of ATLFAST++ (not embedded in the ROOT structure) should be developed.

GEANT4 SIMULATION

-- need extensive comparisons with test-beam data : e.g. shower shapes (lateral, longitudinal), energy response/resolution for electrons and **pions**

-- GEANT3 : hadronic packages (GFLUKA, GEISHA, GCALOR) do not reproduce LAr/ Tile response to $\pi^{\pm}$ (ATLAS-COM-PHYS-99-56)



need a lot of work to understand/tune hadronic physics of GEANT4 (GEISHA)

urgent to have "module 0" simulation for all sub-detectors

-- it would be wise to have another/independent hadronic package (FLUKA): allows comparisons, evaluation of systematics, etc.
Interface FLUKA/GEANT4 in progress (A. Dell'Acqua, A. Ferrari, S. Vanini)

RECONSTRUCTION

Performance/physics groups should:

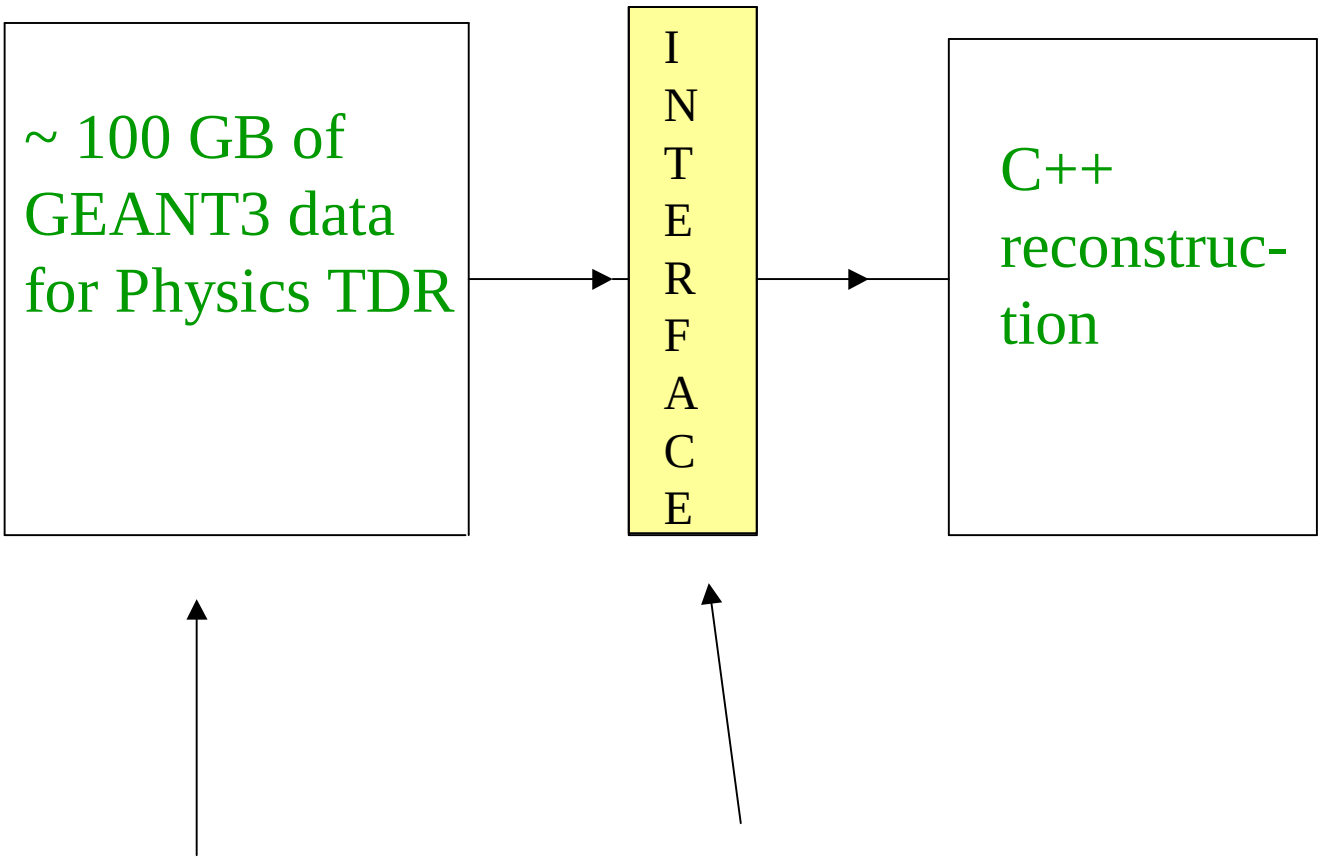
- define requirements and verify that they are satisfied

- define track (ID and muon spectrometer), cluster, ...
  and operations to be performed with them

  Good starting point: definition as in combined ntuple

- contribute to:
  -- reverse engineering of Fortran code
  -- transition to C++/OO
  -- test, test, test … of new code (or pieces of it)
  → performance evaluation

Note : need interface between GEANT3 and new software



need DIGI and HITS

the only huge data sample for long time

GRAPHICS / EVENT DISPLAY

Inputs from performance/physics groups needed. Not much communication in the past.

Only one package could be used for the Physics TDR: PERSINT.

Use of various packages must start as soon as possible → should eventually lead to an evaluation.

ANALYSIS TOOLS

- Code should be independent of analysis/ visualisation package → abstract interface:
  -- don't know today what will be best tool in 2005
  -- different users may want to use different packages

- Could use PAW for many years.
  However: use of more modern and improved tool would help physicists to learn new software techniques (C++, etc.) in "easy" environment

  → ROOT ("PAW-part" only) is good candidate for interim solution: ready to use, well suited to physicist needs
  → discussion needed this week to prepare decision at next CSG and Physics Coord.

- In parallel: evaluation of various packages (ROOT, OpenScientist, JAS, etc.) : involve physics/performance groups; should use combined and ATLFAST ntuples.

Other inputs from physics/performance groups: … a non exhaustive list ...

- contribute to event definition
- contribute to detector description
- understand trigger/event filter requirements
  → implications on reconstruction
- elaborate calibration/alignment strategies
  → implications on reconstruction
- understand event preselection
  → implications on regional centers
- etc. etc.

CONCLUSIONS

Software effort must be driven by physics goals → physics/performance groups can (and are willing to) give significant contribution

Three main requirements (IMHO):

-- use of past experience

-- importance of performance evaluation

-- look for simplicity/easy use

Every package and piece of code must be used used used used by as many people as possible