ROOT Based CMS Framework

Bill Tanenbaum

US-CMS/Fermilab

14/October/2002

CMS Software Framework

- **Objectivity** was used for persistency
- **Objectivity** must be abandoned (long story)
- **ROOT/IO** chosen as component of replacement
- Problem: How to replace **Objectivity** with minimal disruption to CMS software, in a short time, with limited manpower
- Solution: Emulate much of Objectivity with **ROOT**.

ROOT Based Framework

- Replace **Objectivity** with **ROOT** in framework
- All persistency capable classes **ROOT**ified (including metadata)
- Use **STL** classes (e.g. **vector**).
- No **ROOT** specific classes used, except for Persistent References (**TRef** class)
- No redesign of framework
- Foreign classes used extensively

ROOT Based Framework Details

- **Map Objy functionality to ROOT**
  - Objy Database -> **ROOT** file
  - Objy Container -> **ROOT** directory (not Tree/Branch) (Folders not used, either)
  - Objy Ref/Handle -> “enhanced” **TRef**
  - Objy **ooVArray** -> **STL <vector>** (not **ROOT** specific array)

Details cont.

- **Map Objy functionality to ROOT (cont.)**
  - Objy name scopes -> **STL map/multimap**
  - Objy transaction -> Pseudo “commit” (allows run resumption from last “commit”).
  - Objy session -> NONE
  - Objy context -> NONE

Scale of Effort

- One programmer not that familiar with framework
- Removing **Objectivity** references and getting framework to build successfully (3.5 weeks)
- Replacing stubbed **Objectivity** with “equivalent” **ROOT** functionality in framework (3 weeks)
- Rootifying all persistency capable classes, with successful framework & application build (2 weeks)
- Testing/Debugging/Refining (3.5 months)

Current Status

- based on latest Objy. based framework release
- needs ROOT 3.03/09
- currently on its own CVS branch
- will become the main deveopment line this week.
- no more Objectivity based releases!

Performance

- ROOT based framework uses about half the disk space of Objy based framework for typical output (using ROOT compression level 1).
- About 6% run-time increase to most critical task, mostly due to cost of data compression.
- Greater ROOT compression possible with one-time write-time cost (no change in read time).

Persistent Object References

- **TRef uniquely identifies, but does not locate, a persistent object on disk.**
  - ROOT based framework uses a larger class containing additional information (i.e. file name, directory name, object name) to handle this problem. It “works” as long as no file is moved, renamed, etc...
  - POOL will provide a longer term solution

ROOT Impact on Applications

- A ROOT data dictionary must be generated for any persistence capable user-defined class,
- A ROOT data dictionary must be generated for any user defined class used as a template parameter for a persistence capable class.
- Therefore, many application classes must have ROOT data dictionaries generated.
- Therefore, the application and ROOT cannot be totally decoupled.
- ClassDef() not needed- no source code coupling.

ROOT Impact on Apps. (cont.)

- **In generating DATA Dictionaries from header files, not all legal C++ works. It usually works, but:**
  - What doesn't work is not well documented.
  - Much of what doesn't work is not caught by **rootcint**. Rather, the produced dictionary does not compile, or there are link time errors.
  - **rootcint**'s error messages lack detail.
  - Workarounds nearly always possible, but not always obvious.

“commit” without Transactions

- **Purpose**: to be able to restart long runs from last “commit” after a crash or other disruption
- **Solution**: One “collection” object contains refs to other persistent objects.
  - First write out other objects, and write their keys without closing file. Then write collection object.
  - As a “commit”, write the key of the collection object.
  - If crash before the “commit”, all the data since the last “commit” is overwritten. There is no incomplete data!

Parallel Runs without Concurrency

- **Purpose**: to run parallel long jobs without conflict
- **Solution**:
  - First, one short run writes out run independent metadata (a few minutes). No parallelism needed.
  - Then, many parallel event runs write into uniquely identified files. The metadata is accessed read-only. These are the long runs handling event data.
  - An event run can then be attached to the metadata by a separate job taking only seconds.

Summary

- **ROOT** based framework provides a minimally disruptive transition from **Objectivity**.
- Not the final answer: In the near future:
  - Possible internal use of ROOT specific classes (e.g. Trees) for performance/disk space optimization.
  - Possible rework to data model to eliminate residual Objectivity specific optimizations
  - Separation of persistency specifics from framework.
  - Transition to POOL