ROOTifying COBRA

Bill Tanenbaum

US-CMS/Fermilab

06/06/2002

Presentation Outline

- Previous Prototype (March 2002)
- Current Prototype Overview
- Current Prototype Details
- ROOTification Issues
- Future Plans
- Summary

March 2002 Prototype

- Quick **ROOT** Prototype inside **COBRA**
- Expedience over quality
- Don’t disturb **Objectivity**
- Create **ROOT** classes for “**writeHits**”
- **ROOT** classes populated from existing **Objectivity** classes
- It works!

Current Prototype - Overview

- Replace **Objectivity** with **ROOT** in **COBRA/ORCA**
- 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)

Current Prototype Details

- **No Redesign of COBRA**
- **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)

Prototype Details cont.

- **Map Objy functionality to ROOT (cont.)**
  - Objy name scopes -> existing **COBRA** implememtation of multimap using <vector> (DictOfRef)
  - Objy transaction -> NONE
  - Objy session -> NONE
  - Objy context -> NONE

Scale of Effort

- One programmer not that familiar with **COBRA**
- Removing **Objectivity** references and getting **COBRA/ORCA** to build successfully (3.5 weeks)
- Replacing stubbed **Objectivity** with “equivalent” **ROOT** functionality in **COBRA** base (3 weeks)
- Rootifying all persistency capable classes, with successful **COBRA/ORCA** build (2 weeks)
- Debugging (1 week so far - ongoing)

Current Prototype Status

- **ROOT** 3.03/05 **COBRA** 6_0_3/**ORCA** 6_0_1
- Metadata likely to be in Relational Layer also **ROOT**ified for now (easier that way)
- Prototype in debugging stage
- Not supported yet (due to expediency)
  - Remote files
  - External histogramming packages (e.g. HTL)
- Still debugging, no performance/recovery info yet

Major ROOT Issue

- **TRef uniquely identifies, but does not locate, a persistent object on disk.**
  - defined a larger class containing additional information (i.e. file name, directory name, object name) to (poorly) handle this problem. It “works” as long as nothing is moved, renamed, split, etc...

Other ROOT Issues

- **ROOT does not provide unique OID**
  - OID's used by TRef belong to TRef, not to object. If two TRef's refer to same object, they may or may not have the same OID. This complicates, for example, checking if two TRef's refer to same object.
  - Objects not referenced by TRef have no OID at all. (May be OK.)

Other ROOT Issues (cont.)

- **Can’t write all modified persistent objects with single call (i.e., no “commit”)**
  - User must do bookkeeping to know what objects or directories to write, and write them individually.
  - OTOH, the ability to selectively write objects provides flexibility.
  - Opinion: I prefer a commit, even in the absence of ACD transactions.

ROOTCINT

- **rootcint** must generate a data dictionary for any persistence capable user-defined class
- If **rootcint** must generate a DD for class C, it must generate a DD for any user-defined class that is:
  - A base class of C
  - The class of a persistent non-static data member of C
- If B<C> is an instantiated persistence capable class, and C is a user defined class, a DD must be generated for class C.

ROOTCINT (Cont.)

- Because of the above, many classes may need to be rootified, especially if deeply nested templates are used (as in COBRA)
- Opinion: If a templated smart pointer is used, keep a non-templated persistent version available (unless data dictionary bloat is somehow solved/reduced by other means).

ROOTCINT Issues

- **Not all legal C++ works.**
  - 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.
  - **rootcint**’s error messages lack detail.
  - Fixes nearly always possible, but not always obvious.
  - Conclusion: rootcint needs work if the generation of data dictionaries is to be fully automated.
  - Worst deficiency: Too limited support of STL.

Near Future Plans

- Communicate **ROOTCINT** wish list to **ROOT** team (partially done)
- Finish debugging prototype
- Put prototype under source control (CVS)
- Use prototype for tests (Performance, reliability/recovery, scalability, etc.)

Possible Future Plan

- **Persistency implementation dependencies removed from COBRA and placed in separate products.**
- **Implementation independent representations used for:**
  - Persistency capable classes
  - Persistent references
  - Containers/Collections, etc.

Summary

- **COBRA/ORCA** using **ROOT** is running (but still in debugging stage)
- **TRef** needs improvement to locate and read object into memory
- **ROOTCINT** could use improvement to minimize manual user intervention
- Still lots to do to make framework independent of persistency implementation