ROOT Foreign Class I/O in the LCG Framework

Victor Perevoztchikov, BNL

BROOKHAVEN
NATIONAL LABORATORY

The LHC Computing Grid Project - LCG

The LCG project described on http://lhcgrid.web.cern.ch/.

One of the most important part of it is persistency (as usual). The LCG persistency is supposed to be implementation independent. But it is evident, that the ROOT is the main candidate for LCG I/O (if not the only).

Requirements for LCG user API:

- User API is independent from I/O technology;
- The transient user classes also technology independent. They should not be modified or instrumented;
- Initialization of dictionary should be hidden;
- The concrete technology should be on the low level. DB, file catalog should be independent of technology, and vice versa, I/O technology should be independent from DB and file catalog implementation.

Theory

A traditional approach for I/O is a definition of two kinds of classes, transient and persistent:

- Transient classes are user defined and know nothing about I/O technology.
- Persistent classes are heavily dependent from I/O implementation;
- There is a conversion service between these two, provided by user;
- Relationship between the objects (pointers) is supposed to be done by conversion service and usually not discussed;
- Schema evolution is also not discussed too much.

ROOT I/O for Foreign Classes

ROOT I/O basically satisfied to the mentioned theory. But there is some specific:

- There are user defined transient classes;
- There is persistent class, but the only one: TBuffer.
- Conversion between transient classes and persistent one is automatic;
- Relationship between the objects provided by conversion and is also automatic.
- Schema evolution is also part of conversion service and is automatic too. May be it is not too important for industry, but for physical experiments,with classes modified each day, it is extremely important;
- In addition it is not a theory, but working stuff;

So ROOT I/O is a very good candidate for LCG persistency. In my opinion, traditional, one to one, correspondence between transient and persistence classes, in the times of class dictionary is outdated.

Main LCG I/O Classes

- Initialisation: LCGMakeDict class. It is an interface to create the dictionary on fly. Filled by:
  - User class name;
  - Path names to user .h files;
  - Include file names;
  - Etc…

  Method makeIt() produces the dictionary. Internally special makefile invoked, library created (using compilation flags from ROOT) and loaded.

Main LCG I/O Classes (Continued)

- Event: LCGEvt class. Collection of user objects to Read/Write in one transaction. Functionality:
  - Set collection name;
  - Set I/O file name;
  - Set Run,Event number;
  - Add Pointer to object,class name,key name;
  - Or add pointer to object,type_info, key name;
  - Write collection;
  - Read event by given Run/Event numbers;
  - Read next event after current Run/Event numbers;
  - Get object by key or class names;
  - …

Independency

To provide independency of LCG I/O classes from concrete implementation the simplest way is chosen.

Classes LCGMakeDict, LCGEvt, etc… are abstract.

The real classes for ROOT case LCGMakeRootDict, LCGRootEvt, are inherited from the abstract ones.

Standard factory technology could be used to select the needed implementation.

If it will be working non ROOT implementation.

Conclusions

- LCG I/O interface was defined;
- Implementation of interface, based on ROOT foreign classes I/O is created and tested;
- Functionality:
  - Direct access I/O by given Name, Run, Event;
  - Write/Read complicated tree of user objects, non dependent of implementation;
  - Relationship (pointers) between objects is supported;
  - Parallel writing and reading into/from set of files is supported;
  - Schema evolution, supported by ROOT;
- Tests were made with complicated tree of non root objects, including STL vectors and lists.