The CDF Run II Event Data Model and ROOT-Related Topics

FNAL PAS - ROOT Meeting
09 July 1999
Rob Kennedy

I) CDF Run II Event Data Model Prototype
Updated 07 June 1999 Von Ruden Review presentation including:
- **Root Issues related to the CDF EDM (details in e-mail)**
- **CDF's Run II Data Processing Chain (RDK's best guess)**

II) User-Data Interaction within AC++ Framework

CDF Run 1 Event Data Model

Event Data := the data in a physics event record which is passed from algorithm module to module and to/from permanent storage media. No calibration data, geometry, module parameters, etc.

CDF Run 1 Event Data Model := Data fields stored in YBOS banks. Banks labelled by 4 char name and a number assigned by the user. Event record was global array used to store all YBOS banks, internals, etc. YBOS maintained by CDF, few other users.

Common Usage: Get array index for first bank with name "MUON", use index plus offsets in F77 header file to read or write fields of the bank. Get array index to the next bank with same name.

Problems: Corruption of event record when writing with inappropriate offsets, change of an associated object without changing dependent objects, user-defined bank numbers re-used, discomfort with using banks led to unpacking into common blocks....

☞ Model was a success. *But*, can we do better, especially with C++?

Goals of the CDF Run II EDM

- All data passed from algorithm module to algorithm module passed in the event record. No singletons/globals/commons allowed.

- Objects more flexible than YBOS banks can be stored/saved. One is not led to support both transient and persistent forms of objects.

- (Abstract goal) At any algorithm module boundary, we can store the state of the event record, then resume the process later. This must yield the same results as if the process had run to completion.

- Links/associations between objects should be robust. Object identity should be preserved throughout the life of an event record.

- Provide direct support for storable collections of objects in event.

- Abide by the CDF Run 2 Data Handling decision to use ROOT to implement Event I/O, beginning with sequential data files.

- Require as little change to existing C++ source as possible.

Overview of the CDF Run II EDM

➠ C++ classes derived from Edm::StorableObject can be passed in the event record. To be storable on permanent media, a class defines a Root "Streamer()" method (translates transient/persistent form).

➠ StorableObjects are passed by reference using "Handles".
➠ When added to an event, objects are assigned an object_id, unique across the event record for the lifetime of the event record.
➠ StorableObjects use rcp_id for creating module/parameters.

➠ Once in an event, objects are write-locked and cannot be deleted from the event until output is performed.

➠ Links (persistent pointers), StorableCollections are supported.
➠ Storable Banks (YBOS banks in ROOT format) are supported.
➠ Memory mgmt classes used to aid debugging, improve efficiency.

➠ Different ROOT event models (sequential, multi-branch, ...) supported in various EDM File classes. These are not seen by user algorithm code as they are only referred to by AC++ I/O modules.

Sample Use-Cases....

Create object, set a data field (pseudo-C++):

```
Handle<Muon> h_muon(new Muon) ;  // Allocate muon, init handle
h_muon->set_pz(1.0) ;            // Can write into muon now
ConstHandle<Muon> ch_muon = p_record->append(h_muon) ;
// Put muon into event, non-const handle => 0, get const handle
```

Search of object in event record, read a data field

```
ConstIterator iter(ptr_to_event, "Muon") ;  // Find 1st Muon
ConstHandle<Muon> ch_muon2(iter) ;          // Create ConstHandle
string class_name = ch_muon2->class_name() ;// Query possible w/
Id     muon_oid   = ch_muon2->object_id() ; // any StorableObject
float  muon_pz    = ch_muon2->pz() ;     // *Muon*-specific query
```

Very simple Streamer() method (fields are built-in data types):

```
ToyMuon::Streamer(TBuffer& iobuffer)        // Stripped of base class support
{ if (iobuffer.IsReading())                 // Stripped of schema evolution support
    { iobuffer >> _px ; iobuffer >> _py ;   // Stripped of error/sanity checks
      iobuffer >> _pz ; iobuffer >> _energy ;
    }
  else if (iobuffer.IsWriting())
    { iobuffer << _px ; iobuffer << _py ;
      iobuffer << _pz ; iobuffer << _energy ;
    }
}
```

Current Status

- Prototype Release: Complete EDM design and unoptimized implementations of all supporting elements of the EDM. Some "toy" versions of high-level physics objects implemented as tests and to demonstrate application of EDM. Announced 06 July 1999.

- ROOT event model: SeqRoot = A single TTree containing a single TBranch containing Event Record objects. I/O performance testing done, retested at regular intervals. A multi-branch event model early prototype exists (Pasha Murat, Lena Gerchtein), not yet in EDM.

- Improvements in ROOT: have been requested of the ROOT team to support the CDF Run II EDM (list on later slide). No show-stopper issues, but some work-arounds have had to be used.

- High Priorities in Short-term: I/O Modules for use of EDM in AC++/Framework, documentation and presentations for CDF Week (26-30 July), Namespace support in EDM-Root, "high-level" physics object classes like Track, Muon, Jet, MissingEt, etc.

Plans for Future

After the Prototype Release

- Begin evaluation of new EDM by Physics groups. Finalize high-level physics objects. Easy to code once agreement on content.

- With Physics groups: define a transition plan to agreed to production EDM. May require a "composite" YBOS/EDM2 record.

- With Physics groups: define Fortran <=> event data policy. Event data accessible to F77 in YBOS banks, via commons, via arg lists?

- (Underway) Upgrade to use ROOT v2.22.06 for CDF Offline dev. Taking much effort, unlikely to be upgraded again soon.

- Develop more efficient, robust implementations of EDM, including smarter handles/pointers, memory managers, ...

- Production Release: Expected in Summer 1999, contains all above plus "getting started", "making the transition with existing code".

EDM Requests to ROOT Team

- C++ Namespaces: The ROOT I/O mechanism does not support user-defined namespaces. Example has been provided, but no estimate yet. Trying w/new ClassDef/ClassImp, then rootcint fails.

- Rootcint/Streamer Support for Stdlib Classes: STL, string, bitset. STL is supported now. We are hand-coding our Streamer()s, so this is not holding us up. We define own operator <<, >> for std::string.
- Improved Rootcint Robustness: rootcint does not correctly treat some C++ constructions related to nested classes, incomplete types.
- Byte Count Mechanism: Already implemented in ROOT v2.22/06. We request some hooks to pass errors to our error logging package.

- Link-time dependencies: Edm can now use "dummy" versions of libPostScript, libMinuit, and libGraf3d safely. Topic dropped.

- Miscellaneous: Improved CPU efficiency for sequential event model (are we using ROOT as efficiently as possible?). Hooks to permit better integration of ROOT "split-file" mode with CDF Data Handling system (CDF needs to define better).

CDF Run II Data Processing Chain

➠ ***Caveat***: Plans for the complete data processing chain are not finalized. They are dependent on many factors and decisions. This is ****only**** one of several possible scenarios under discussion.

➠ Raw data is read out of the detector into the Level 3 Trigger processes as YBOS banks (compact, efficient storage of raw data).
➠ These are translated into StorableBanks in the input module to the Level 3 reconstruction program, which fully utilizes the new EDM. It outputs StorableObjects in a sequential, uncompressed ROOT event. Tape hardware compression may be used.

➠ Production and on-line consumers receive this sequential ROOT data as input. The production executable uses the new EDM, outputs sequential, uncompressed ROOT events into many data streams.

➠ Secondary and tertiary data sets may benefit from using multi-branch or split-file ROOT events, depending on use patterns. Objects are collated on input/output into Branches per a mapping. The mapping is treated as meta-data and maintained in database,

User-Data Interaction in AC++

The Big Picture

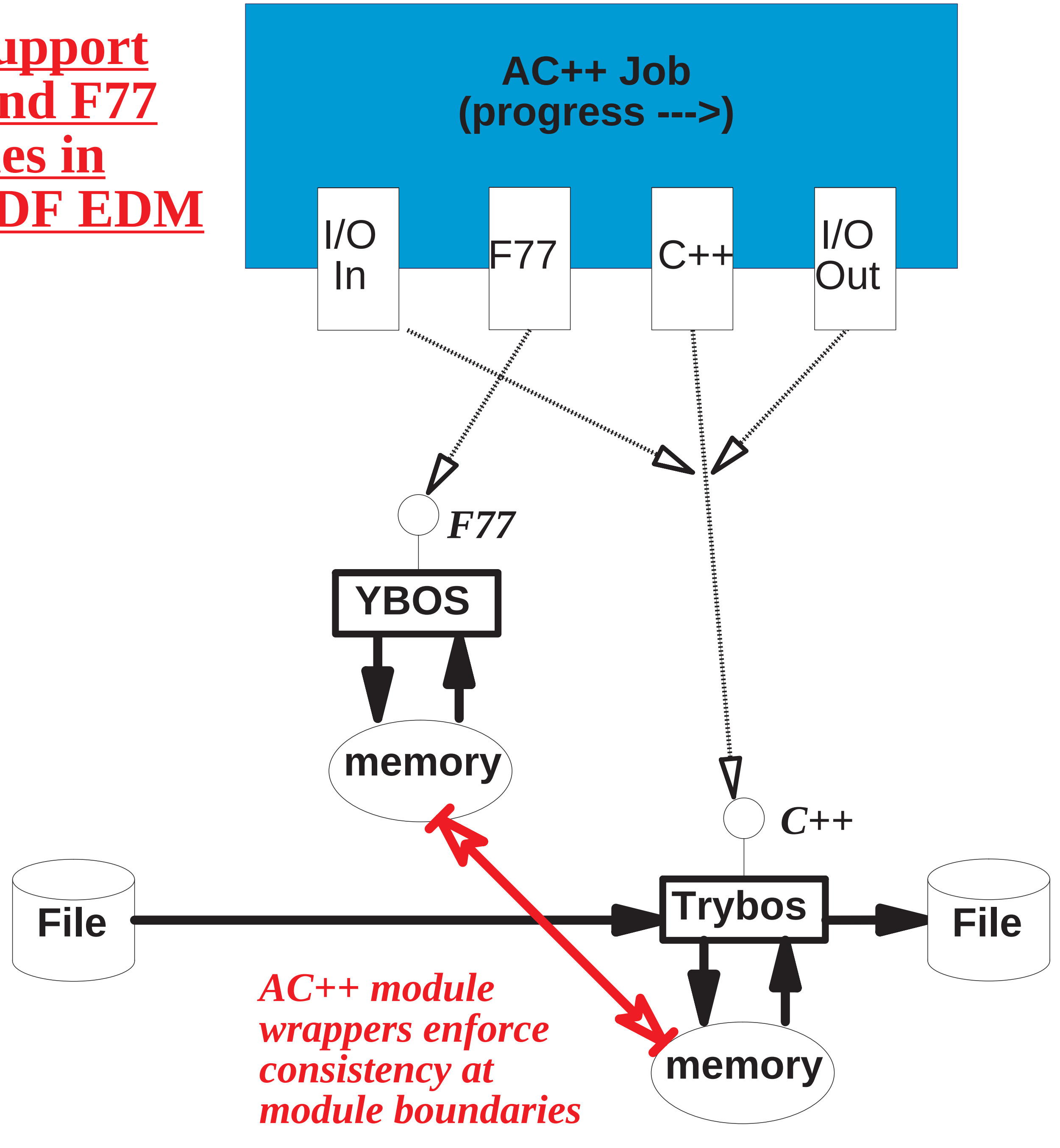
- **Framework/AC++: A jointly developed (BaBar/CDF) package to support in C++ the development of large-scale analysis programs modelled as a particular finite-state machine. AC++ Project page:**
http://cdfsga/upgrades/computing/projects/framework

- **Users create *module* classes with *entry points* that are called at the appropriate place in the program's state diagram, for example BeginJob(), BeginRun(), Event(), EndRun(), EndJob(), etc.**

- **I/O modules: the only modules to input/output event data. Users select which I/O module is appropriate for their task or datafile. I/O modules exist for YBOS diskfiles, Level 3 shared memory, etc.**

- **Big complication: Data access required by wrappered Fortran modules. Declare each module to use either C++-style data access module (preferred) or Fortran-style data access module. Trybos (C++ re-invention of YBOS) uses memory layout similar to YBOS. Copy event data: Trybos <=> YBOS when data access style changes.**

AC++ Support of C++ and F77 Modules in Existing CDF EDM



User-Data Interaction in AC++

The Small Picture

- Hooks to the Event Data: Each module's Event() entry point has access to the event data via AbsEvent*. For CDF, this is macro for a pointer to a Trybos event.
- Hooks to the "Environment": A small set of globals are maintained: run number, trigger/event number, etc.

- I/O Modules: User states which data files to use as parameters to the appropriate I/O module. New EDM: I/O module to input Ybos, translate to StorableBanks, put into EventRecord.

- C++ Data Access: Users create specialized iterators into the Event Record which apply some selection criteria to search for event data elements of interest. Old EDM: select by bank name, bank number, or no criteria. New EDM: select by class name, object id, rcp id, or a general description (a user-defined string in data).

- C++ Data Elements: Old EDM: YBOS Bank instances. New EDM: StorableObjects. Generally composite with mixed data types.