The Design and Status of the New eGamma Framework

Ricardo Goncalo, Tania McMahon and
Mark Sutton

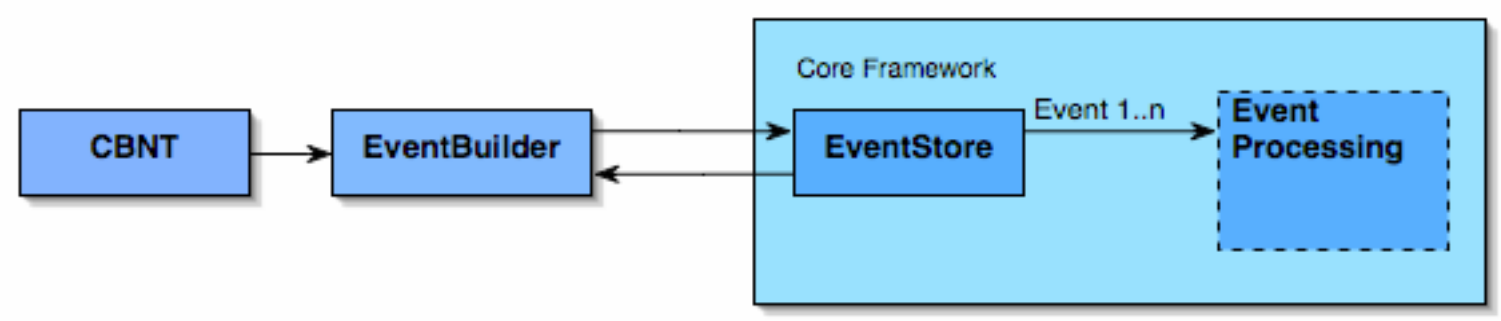
29th June 2005

Introduction - Why a new framework?

- Old framework, not easily maintainable …
  - Difficult to add new signatures.
  - Multi object triggers a problem.
  - Difficult to modify cuts (many are hard coded).
  - Dependence of algorithm code on CBNT structure.
- New framework …
  - Factorise the CBNT storage from the core Framework.
  - Internal representation of event structure and classes closer to those used online,
    - Should facilitate quick porting of hypotheses algorithms to and from Athena framework.
  - Fast, lightweight framework,
    - work started interfacing with automated trigger optimisation strategies.
  - New signatures are be simple to implement.
  - Multiple object triggers are no more difficult than single object triggers.

Code structure - event storage

- FrameworkEvent
  - Lightweight internal representation of the event, organised by RoI, has track collections, Calo Clusters.
  - Internal classes designed to be similar to online classes.
- EventStore



  - Responsible for accessing the persistent storage (ie CBNT) and building each FrameworkEvent
  - Factorises event storage from event processing.
  - Can be used to read events sequentially from the file for processing, or buffer all events in memory for speed (automated tuning strategies).

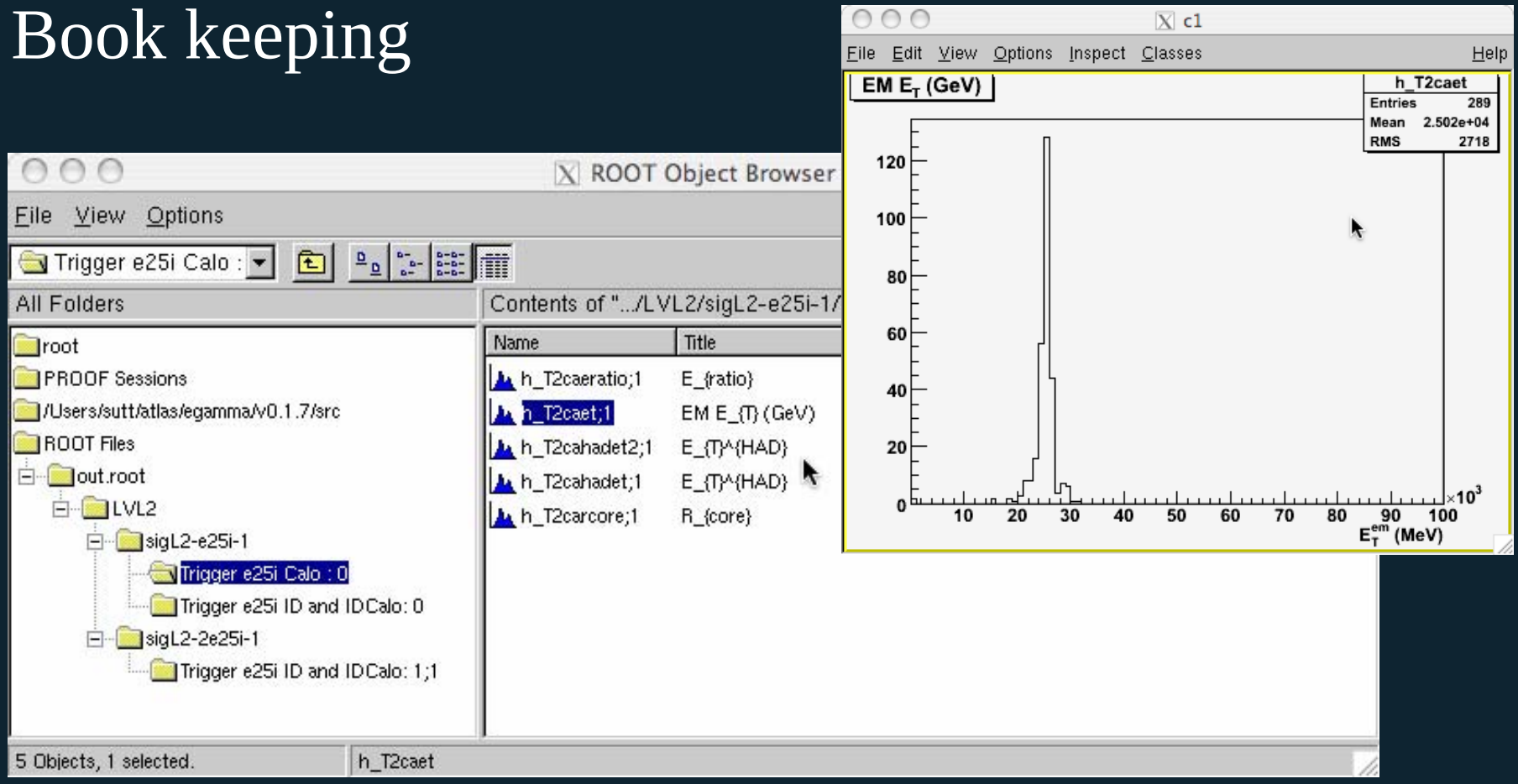
Trigger classes

- LVL1, LVL2 and EventFilter
  - Each contains a collection of TrigSignatures, each with it's own prescale.
  - The output of all TrigSignatures are OR'ed together to get overall decision at each trigger level.
  - Does their own book keeping, number of events passed, failed etc.
  - Monitors event correlations between TrigSignatures, overlap of numbers of events of each TrigSignature with all others etc.

TrigSignature and TrigSequence

- TrigSignature
  - Contains some number of TrigSequences.
  - Runs each TrigSequence on all RoI's which matches its required input type.
  - After all apporiate sequences have been run on each RoI, have a vector of TriggerElements that are examined by the TrigSignature, to search for the required combination of objects.
  - All book keeping (number of events passed, before/after presecale etc) done automatically by the TrigSignature.
- TrigSequence
  - Event book keeping of the TrigSequence done automatically.
  - All other code, isolation cuts, matching, histogram filling etc, done as required by the user in the derived class.

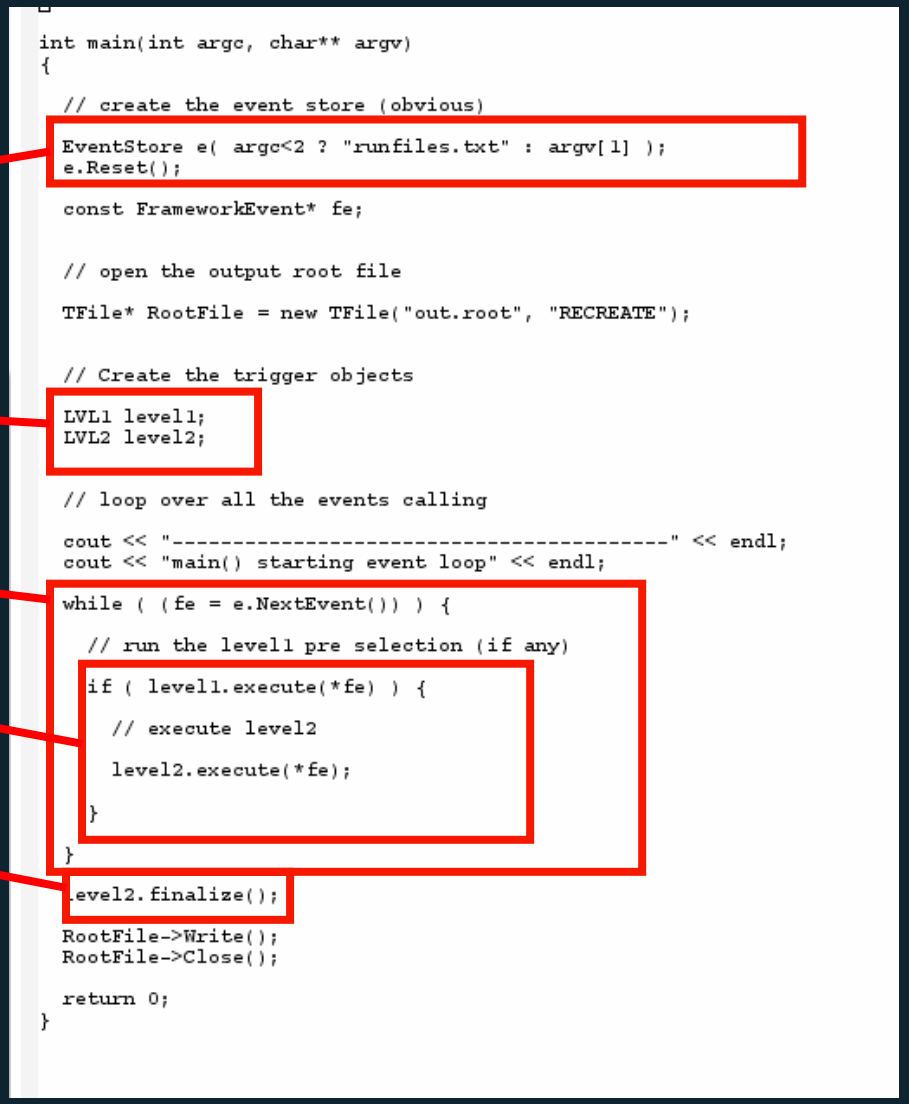
Book keeping



- All objects can configure themselves from their own configuration file (or from a single master file if required).
- All objects handle their own book keeping.
- Histogram output automatically organised using root subdirectories.

Concrete example

- EventStore
- LV1 and LVL2
- Event loop
- Execute algorithms
- Finalise algorithms



```
int main(int argc, char** argv)
{

  // create the event store (obvious)

  EventStore e( argc<2 ? "runfiles.txt" : argv[1] );
  e.Reset();

  const FrameworkEvent* fe;


  // open the output root file

  TFile* RootFile = new TFile("out.root", "RECREATE");


  // Create the trigger objects

  LVL1 level1;
  LVL2 level2;


  // loop over all the events calling

  cout << "----------------------------------------" << endl;
  cout << "main() starting event loop" << endl;

  while ( (fe = e.NextEvent()) ) {

    // run the level1 pre selection (if any)

    if ( level1.execute(*fe) ) {

      // execute level2

      level2.execute(*fe);

    }
  }

  level2.finalize();

  RootFile->Write();
  RootFile->Close();

  return 0;
}
```

Concrete example - LVL2 e25i

- Implement class TrigSignature_L2_e25i.
  - Specify which sequences should be run, and how they should be combined.

```
TrigSignature_L2_e25i::TrigSignature_L2_e25i(std::string name) : TrigSignature(name) {
  cout << "TrigSignature_L2_e25i::TrigSignature_L2_e25i(): " << Name() << endl;
  mDir.push();
  mSeqVec.push_back(new TrigSequence_L2_e25i("Trigger e25i : 0", "sequence_L2_e25i.txt") );
  mSeqID.push_back(mSeqVec[0]->GetID());
  mDir.pop();
}
```

  - The user sequences implement hypotheses algorithms. These do most of the work, user decides (and codes) which variables to be used, which cuts to apply.
  - Custom bookeeping (eg sequence specific histogramming) performed by derived class.

- Standard book keeping, (event counting) performed by the base classes for TrigSignatures and TrigSequences

Concrete two object example - LVL2 2e25i

- Once the single object sequences have been defined, implementing 2 object triggers is trivial,

```
TrigSignature_L2_2e25i::TrigSignature_L2_2e25i(std::string name) : TrigSignature(name) {

  cout << "TrigSignature_L2_2e25i::TrigSignature_L2_2e25i(): " << Name() << endl;

  mDir.push();

  mSeqVec.push_back(new TrigSequence_L2_e25i("Trigger e25i: 1", "sequence_L2_e25i.txt") );

  mSeqID.push_back(mSeqVec[0]->GetID());
  mSeqID.push_back(mSeqVec[0]->GetID());

  mDir.pop();

}
```

- Again, all book keeping handled automatically.

Sample performance

- Timing performance …
  - EventStore, approximately 150 events per second[1] can be read and constructed.
  - With coded LVL2 e25i and LVL2 2e25i, around 7000 events per second[1] can be processed,

```
main() starting event loop
main() event loop execution time 43.647 ms   300 events   43.65 ms
LVL2 events 300  passed 250  effciency for cut LVL2 = 0.833333 +- 0.0215166
TrigSignature::finalize():sigL2-2e25i-1:  efficiciency 0 +- 0
TrigSequence::finalize():Trigger e25i: 1:  efficiency for counter Trigger e25i: 1 = 0.862069 +- 0.020249
...
...
...
TrigSignature::finalize():sigL2-e25i-1:  efficiciency 0.833333 +- 0.0215166
TrigSequence::finalize():Trigger e25i : 0:  efficiency for counter Trigger e25i : 0 = 0.862069 +- 0.020249
TrigSequence::finalize():Trigger e25i : 0:  total for counter      Trigger e25i : 0 = 290
TrigSequence::finalize():Trigger e25i : 0:  passed for counter     Trigger e25i : 0 = 250
TrigSequence::finalize():Trigger e25i Calo : 0:  efficiency for counter Trigger e25i Calo : 0 = 0.941379 +- 0.0137946
TrigSequence::finalize():Trigger e25i Calo : 0:  total for counter      Trigger e25i Calo : 0 = 290
TrigSequence::finalize():Trigger e25i Calo : 0:  passed for counter     Trigger e25i Calo : 0 = 273
TrigSequence::finalize():Trigger e25i ID and IDCalo : 0:  efficiency for counter Trigger e25i ID and IDCalo : 0 = 0.915751 +- 0.0168109
TrigSequence::finalize():Trigger e25i ID and IDCalo : 0:  total for counter      Trigger e25i ID and IDCalo : 0 = 273
TrigSequence::finalize():Trigger e25i ID and IDCalo : 0:  passed for counter     Trigger e25i ID and IDCalo : 0 = 250
```

NB. Single 25GeV electron Monte Carlo

[1] timed on a 1.5 GHz G4 PowerBook.

Status so far …

- Essential core structure is in place and usable
  - Read events to the EventStore,
  - Can execute arbitrary numbers of Signatures each with an abitrary number of Sequences with arbitrary combination schemes, ie multi object triggers
  - An implementation of prescales included.
  - Resembles online steering logic, facilitates porting of code to and from Athena framework.

- User code, concrete LVL2 e25i single object and LVL2 2e25i double object triggers coded,
  - no more difficult to implement multiple object triggers than single object triggers.
  - Basic book keeping is done automatically.

Outstanding tasks and additional issues

- Outstanding tasks …
  - Implement pre-filtering on truth information,
  - Implement Event Filter signatures,
  - Improve communication between trigger levels,
  - Begin detailed validation (should start within one or two weeks),
  - Integrate with automated optimisation schemes (multiple event stores).
- Desirable external requirements or requests …
  - Muons for b tagging hypotheses in the CBNT,
  - RoI information in the CBNT (which objects, tracks, clusters etc, belong to which RoI)
  - Jet information?
  - Any other requirements?