Algorithms & EDM
Exercise using a Z to ee algorithm in Athena

S. Rajagopalan

Outline

The Athena Transient Data Store a.k.a. StoreGate (SG)

How algorithms interact with this store

Writing two algorithms:

- **A Z to ee algorithm that builds Z objects from previously built egamma candidate objects (output of standard Reconstruction)**
- **A Z to ee CBNT algorithm that fills an ntuple with the computed Z to ee variables.**

StoreGate

The ATLAS Transient Data Model Infrastructure

To manage Data Objects life-time

- SG manages data objects memory (owns them)
- SG interacts with the persistency to read/write data objects.

To access Data Objects

- Type-Centric Naming service

  Give me the TrackCollection called “MyTracks”
- Navigation

  Persistable references (DataLinks and ElementLinks)
- Memory Management Tools

  DataPtr, DataVector, DataList
- History

Objectives

Learn how to access data objects using StoreGate

- Locating event and detector store
- Configuring your data objects

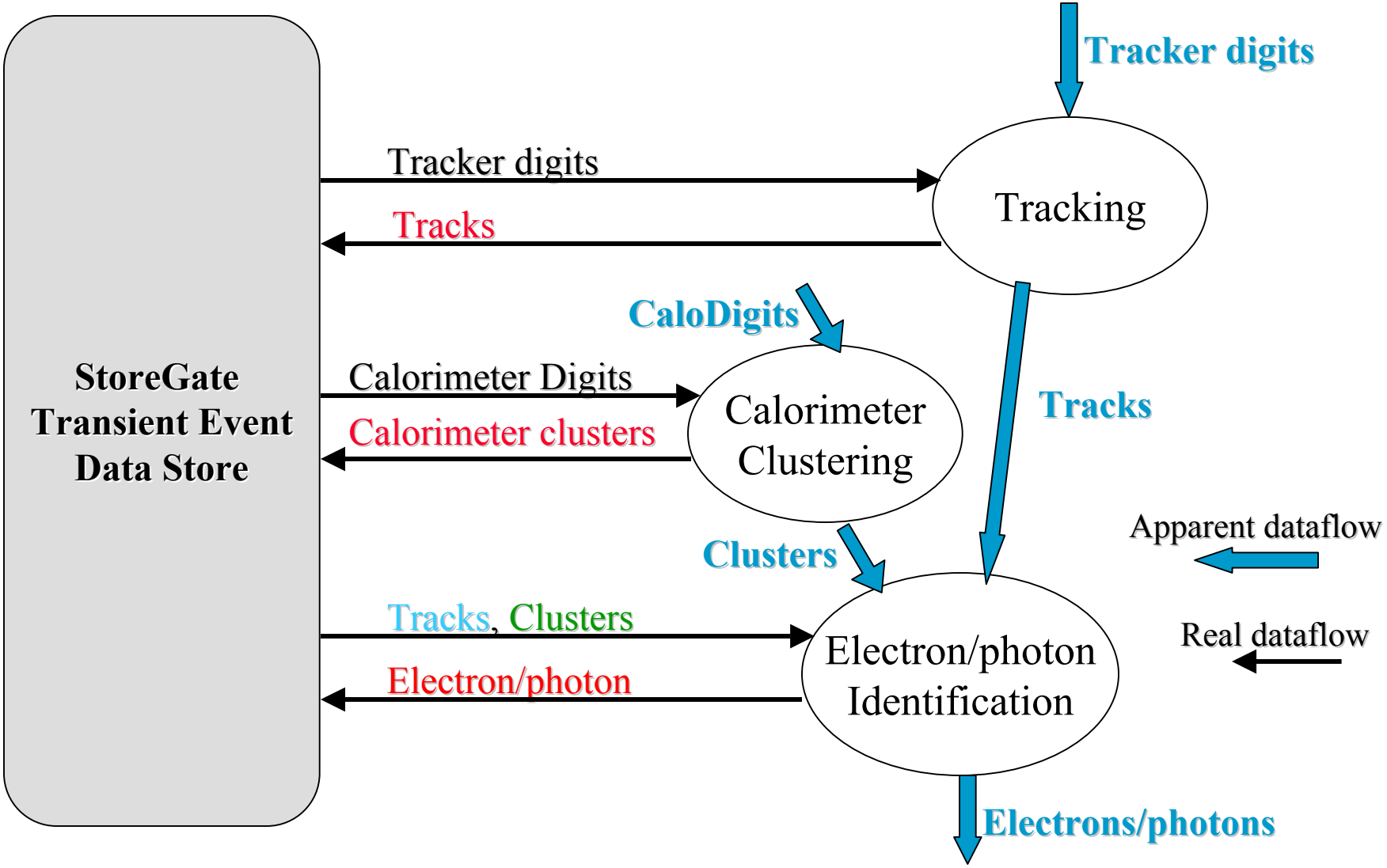
  how to use of memory mgmt tools
- How to record/retrieve by TYPE

  Optionally using keys
- Retrieving of all data objects of a given type
- How to use Data Links

  Persistable relationships

This will be done in the context of writing a Z to ee analysis algorithm in Athena.

The Data Flow



Accessing StoreGate

StoreGateSvc is the Gaudi service that allows to record and retrieve data objects.

In the initialize method of your algorithm:

```
StatusCode sc = service("StoreGateSvc", m_storeGate);
```

declare the event store pointer, m_storeGate as private data member of type StoreGateSvc*

This caches m_storeGate and and you do not have to call the serviceLocator to retrieve the pointer to StoreGateSvc every event in your execute method.

If you need access to the detector store as well, add:

```
StatusCode sc = service("DetectorStore", m_storeGate);
```

What is a Data Object?

A Data Object is an object recorded to StoreGate

- and StoreGate is a store of Data Objects…

Most C++ object can be stored to SG and hence are Data Objects

- Must provide a default and a copy constructor (if you are lazy the compiler will provide them for you)
- Must be associated to a magic number: the CLID

A Data Object is created on the heap

```
ZeeContainer *pZeeCont (new ZeeContainer);
```

A recorded Data Object is owned by StoreGate

- NEVER, EVER delete a data object after it has been recorded!!!

```
m_storeGate → record(pZeeColl, myKey);
//DON'T delete pZeeColl!; //SG will do it for you
```

The CLID Thing…

Take a look at ZeeCollection.h

```
typedef std::vector<ZeeObject> ZeeContainer;
CLASS_DEF( ZeeContainer, 9903, 1 )
  //“9903” is the famous CLID, used by persistency
  //“1” is the version number (currently ignored by SG)
```

Every data object in StoreGate carries a CLID

ZeeContainer has a CLID, but Zee object does not

CLID must be unique through Atlas

CLIDSvc and associated scripts:

- provide new CLIDs for new classes
- check uniqueness

Which Collection?

Most Data Objects are Collections OF

- LArDigits, Tracks, egamma, …

Prefer to use STL containers whenever is possible

```
typedef std::vector<ZeeObject> ZeeContainer;
```

- Simple, optimized, no memory mgmt issue, standard
- Unfortunately don't marry well with Abstract Classes

Use Atlas DataVector and DataList when you must

- CaloCell is the base class (ABC) for all calorimeter cells

```
typedef  DataVector<CaloCell> CaloCellContainer
```

- CaloCellContainer behaves like a vector<CaloCell*> that owns its CaloCells

Recording an Object

Providing a Key:

```
static const bool SETCONST(false);
StatusCode sc =
    m_storeGate → record(pZeeCont,  m_ZeeContName, SETCONST);
```

where :

pZeeCont is a pointer to your ZeeContainer (a DataObject)

m_ZeeContName is an identifier for your data object also specified in your jobOptions in a similar way. It can be a simple string like: "MyZeeCont"

Locking an Object (if not done during record)

```
StatusCode sc = m_storeGate → setConst(pZeeCont);
```

Retrieving an Object

Here is an example on how to retrieve a specific data object:

```
const ZeeContainer*  pZeeCont;  // Note the const!
sc = m_storeGate → retrieve(pZeeCont,  m_ZeeContKey);
ZeeContainer::const_iterator fZ(pZeeCont → begin());
ZeeContainer::const_iterator eZ(pZeeCont → end());
// The following sums the et of the Z objects
while (fZ != eZ) { etTot += fz++ → et() }
```

Store Access Policy

An object in SG may be modified until it is setConst

Access to a const Data Object:

```
const TrackCollection*  pTracks;
m_storegate → retrieve( pTracks,  key );
```

- Only const methods in the Data Object are accessible!

```
class Track {
    void set_pT(float pT)    {  m_pT = pT;  }       //  NO ACCESS
    float get_pT    const    {  return m_pT; }      //  ACCESS OK
    …                                              }
```

- Must use const-iterators to iterate over TrackCollection

If you do not specify const, this will force a check. If the data object you are accessing is 'const', then an error is returned if accessing it in a non-const way.

Retrieving all Data Objects of Given Type

If you have several TrackCollection instances in SG, here is an example on how to retrieve ALL of them:

```
const DataHandle<TrackCollection>  dbegin,  dend;

m_pEvtStore → retrieve(dbegin, dend);

for (;  dbegin != dend;  ++ dbegin)        // loop over TrackCollections
{
    dbegin → method_in_trackCollection();
    TrackCollection::const_iterator   iter   dbegin → begin();
   for (;  iter != dbegin → end();  ++iter)      // loop over TrackObjects
    {
          (*iter) → TrackPT();                  // call some method in Track
    }
}
```

EDM Documentation

a) StoreGate Guide (for architectural description)

b) The ATLAS Raw Data Model Document

Linked from the ATLAS EDM web page:

- http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/EventDataModel

c) The Reconstruction Task Force Document (RTF)

- Linked from http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/domains/Reconstruction

Exercise

In Reconstruction/RecExample/ZeeRec/src, you will find:

- ZeeBuilder.cxx
  - This is the top algorithm that will find the Z's.
- CBNT_Zee.cxx
  - Top Algorithm that will fill the ntuple with Z parameters
- DataClasses:
  - ZeeObject
    - an instance for each Z candidate found
  - ZeeContainer
    - a collection of Z's recorded in StoreGate

The header files, properties and variable initialization is already done for you. Use these.

Overview of the Exercise

The standard ATLAS reconstruction constructs egamma candidates. These are available in StoreGate.

We will write a ZeeRec algorithm:

- **that retrieves collection of egamma candidates from StoreGate**
- **Loops over the egamma candidates in this collection and selects a subset of these candidates using some cuts.**
- **pair's them and forms Z to ee candidates (Zee Object)**
- **if they are within the mass window, pushes the zee candidate into a Zee collection.**
- **Records the Zee collection in StoreGate.**

ZeeBuilder

a) Retrieve the const egammaContainer from StoreGate. The egammaContainer key is m_egContainerName. Let us call this collection egColl.

```
const egammaContainer* egColl;
StatusCode sc = m_storeGate->retrieve(egColl, m_egContainerName);
If (sc.isFailure()) {
    mlog << MSG::ERROR << " Error retrieving egamma Container "
    << endreq;
    return sc;
}
```

b) Create a temporary vector of egamma (and other initialization):

```
std::vector<const egamma*> myEGcontainer;
```

Some other variables we are going to use are already initialized:

```
const egamma* eg = 0;              // pointer to egamma object
const LArCluster* clus = 0;        // pointer to LAr Cluster object
const EMShower* shower = 0;        // pointer to an EM Shower object
int nEgHighPt = 0;                 // number of high PT egamma candidates
```

ZeeBuilder (2)

c) Loop over the egamma's in the container:

```
egammaContainer::const_iterator itr = egColl->begin();
egammaContainer::const_iterator iend = egColl->end();
for (; itr != iend; itr++) {
    eg = *itr;                  // Note: eg is already defined for you!
     // <- here we can apply cuts and select the eg's (step d,e,f,g)
}
```

The egamma objects (eg) has a pointer to various objects:

- The pointer to the calorimeter cluster (clus)
- The pointer to a ShowerShape object (shower)
- The pointer to the Track (if one found) (not used in this exercise)
- The pointer to a Track Parameter object. (not used in this exercise)

There are accessor methods for these in the egamma object.

ZeeBuilder (3)

INSIDE THE PREVIOUS LOOP of step c):

d) access the pointer to the cluster and shower shape objects, using the accessors in egamma object

```
clus = eg->get_Cluster();          // Note: clus is already defined for you
shower = eg->get_EMShower();  // Note: shower is already defined
```

e) Check if clus and shower are non-zero (valid).

```
if (clus != 0 && shower != 0) { …
```

f) If so, check if the cluster ET is > m_electronPtMin

```
if (clus->et() > m_electronPtMin) { …
```

g) If so, push back the egamma object in your list

```
myEGcontainer.push_back(eg);
} } // close brackets for the two if conditions
```

ZeeBuilder (4)

If you finished a-g, you have successfully accumulated a list of egamma that pass the minimum PT cut in your private egamma container (myEGcontainer)

Now you will have to match the two egamma objects that give you the Z. Note that you could get lucky and find more than one Z: hence you have to pair all possible egamma objects and check if any are in the Z mass window.

For the sake of the exercise, we will ignore requiring a track associated with the egamma. We only required a simple PT cut.

ZeeBuilder(5)

g) Create a ZeeContainer, here you will collect all the Z's you will find. Already done for you!

```
ZeeContainer* theZeeContainer = new ZeeContainer();
```

h) Check how many egamma objects you have:

```
n_eg_highPT = myEGcontainer.size();
```

i) Check if n_eg_highPT >= 2 (2 is set via property!)

ii) Make a double loop over your private list of egamma's:

iii) Create a Zee object and pass the two egamma's to the Zee

iv) Check if the Zee mass is in window

v) If yes, push back the Zee in your container and print message

ZeeBuilder (6)

```
If (n_eg_highPT >= 2) {
      int I_eg1 = 0;
       int i_eg2=0;
       for ( i_eg1=0 ; i_eg1<n_eg_highPt-1 ; i_eg1++) {
          for ( i_eg2=i_eg1+1 ; i_eg2<n_eg_highPt ; i_eg2++) {
         // Create a Zee object:
             ZeeObject zee ;
         // Pass the egamma object to the Zee object
             zee.setDaughters(theEgContainer[i_eg1],theEgContainer[i_eg2]);
       // Check if the Zee is within the mass window
             if ( zee.mass() > m_ZMassMin && zee.mass() < m_ZMassMax ) {
             // put the found Zee in Zeecontainer
             theZeeContainer->push_back(zee);
            // Print out a message that you have found a candidate
              mlog << MSG::INFO << "Candidate " << i_eg1 << i_eg2 << " mass "
                   << zee.mass() << endreq;
           } } } }
```

ZeeBuilder (7)

Now, you have your ZeeContainer.

Record it in StoreGate.

```
sc = m_storeGate->record(theZeeContainer, m_ZeeContainerName);
if (sc.isFailure()) { // print an error message and return sc }
```

That is it, you are done.

Now, you could do the same exercise for combining truth electrons and forming a Z object. Then you could record the ZeeTruthContainer also in StoreGate and compare the results of the two containers.

CBNT_Zee

The CBNT file does the following:

- In the initialize method, it creates an ntuple and adds items to this ntuple.

In the execute method:

- It retrieves the ZeeContainer from StoreGate that you recorded in the ZeeBuilder algorithm.
- It loops through all the Zee
- It extract relevant parameters from the Zee object
- It fills the ntuple

All this is done for you… But commented out. Look through it and uncomment code.

Zee JobOptions

Zee jobOptions controls all the parameters used in the Zee algorithms. It currently includes:

- **Name of the shared library**
- **Name of the Top Algorithm to be executed**
- **Specific properties:**
  - **Name of the input egamma Collection**
  - **Name of the output Zee collection**
  - **Minimum PT of electron**
  - **Maximum eta of electron**
  - **Z mass window**

To change the parameters, cd to the package share area:

- **Reconstruction/RecExample/ZeeRec/…/share**
- **Edit (emacs) the jobOptions and change parameters.**

- **You can change these parameters, add additional parameters and re-run the job**