Object Oriented Tracking Software for the ATLAS Experiment

Julius Hřivnáč

Institute of Physics, Academy of Sciences
Na Slovance 2, Prague 9
Czech Republic

MOOSE Collaboration

CERN

After about one year of existence, the CERN Project MOOSE[1] (RD41) already has its first running prototype of the Object Oriented tracking software for the ATLAS detector - **Prototype II**. General questions about the MOOSE project[2] and CMS prototype[3] are covered by other talks.

1 ATLAS Collaboration

ATLAS[4,5] is one of two major experiments, which will be built in CERN on the LHC.

The LHC experiments will be **bigger** and **more complex** than current HEP experiments, in detector size and complexity, number of physicists, event size, bunch crossing rate and almost certainly experiment lifetime. It is therefore crucial that all elements of an experiment should be designed, built and maintained to the **highest engineering standards**. **Correctness of software** has not been a major issue until now. It must be taken very seriously at LHC because complex software will be part of each LHC experiment's trigger.

2 MOOSE Collaboration (*ATLAS part*)

Software development for the LHC experiments has already started using methods traditionally used in HEP for many years. The experience from this software indicates that the software to be used during LHC operation should be done on a different basis.

The MOOSE Collaboration (RD41@CERN) has been established to study the viability of the Object Oriented approach for software development for High Energy Physics reconstruction and analysis code at the LHC.

MOOSE:

- use **Object Oriented** technology.
- follow **Software Engineering** methods.
- are implementing the algorithm to **recontruct tracks** in the silicon detectors.
- programme in the **Eiffel** language.
- use **Objecteering** and **OMO**[6] CASE Tools.
- use **cvs** for code management.

3 Prototype II

3.1 Problem Statement

The project is to identify electrons in the ATLAS detector. To do this, following steps are to be performed:

1. The cluster finding in the EM calorimeter. Found clusters then serve as seeds for the track finding in the Inner Detector.
2. The track searching using a road to collect position measurements (called clusters) in the different detector elements (called media).
3. The track reconstruction from the set of position measurements which fit an electron hypothesis (helix with a possible radiation).

3.2 Goal

The final goal of this project is to reconstruct electrons in the ATLAS EM Calorimeter and Inner Tracker. Several steps should be performed for that:

1. To read geometry definition of the ATLAS detector and build its representation in OO language.
2. To read Calorimeter and Tracker data simulated by current ATLAS software (Geant based).
3. To find EM Calorimeter Clusters, which will serve as Inner Tracker Seeds.
4. To find Tracks in Inner Detector.
5. To write all useful data as Persistent objects.

Two simplifications have been made for the first phase of the work:

- In the first implementation only the barrel is considered, in a second round the model will be extended to the forward and backward region.
- In the tracker only the Silicon detectors is used, the Transition Radiation Tracker TRT will be added in the next stage.

The MOOSE environment is covered in other talks. This talk will concentrate on the Prototype itself.

3.3 Design

The main stream of work concerns the analysis, design and coding of the tracking part of the program. Starting from the Problem Statement, following schemas have been designed:

- **Geometry** and **Media** for the detector description.
- **Input**, **Events**, **Raw-Data** for input and steering of data.
- **Trees** for organising and access of the data.
- **MOP** to add Persistency to objects.
- **Calorimetry** for defining a seed, which serves as one of the starting points for the definition of the Road in which Pattern Recognition is performed.
- **Tracking** for implementing of the tracking algorithm itself.
- **Helix-fit** for fitting the track parameters and judging the quality of potential hit association.
- **Circular-Pattern-Recognition** for the recognition of more complicated sets of tracks.
- **Math**, **IO**, etc. as basic service class libraries.

Some of these schemas will be now described in more details. Class-names will be generally typed in **BOLD CAPITALS**, Schema-names will be in **Bold**.

Geometry and Media

- **Geometry**(Fig. 1) contains volume classes.
- **Media**(Fig. 2) contains the detector hierarchy (**ATLAS**, **ECAL**, **INNER_DETECTOR**, **SUPER_LAYER**, ...).

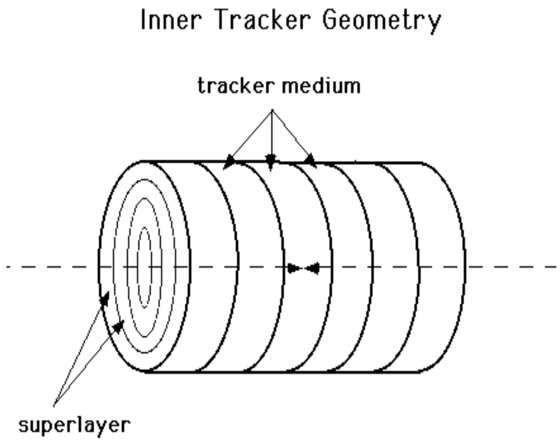

Figure 1: Inner Tracker Geometry.

Inputs, Events, Raw-data

- **Input** contains the classes needed to read the ASCII input file.
- **Events** does top level control, like: initialisation of the detector hierarchy (from file or dbase), and loop over events (from file or dbase), for each event: trigger detector hierarchy and reconstruct event.
- **Raw_Data** contains the persistent classes which hold the raw data from the input file.

Trees

The detector hierarchy is a structure used for the reconstruction of all events. We need to provide access to:

- the geometrical information of the different detector parts,
- the data per subdetector for each event, (one can ask **ECAL** for its energy clusters and **TRACKER_MEDIA** for their position clusters),
- the tracker hits by grouping them:
  - in radius (**SUPER_LAYER**),
  - along z (**TRACKER_MEDIA**),
  - and in phi (**CLUSTER**).

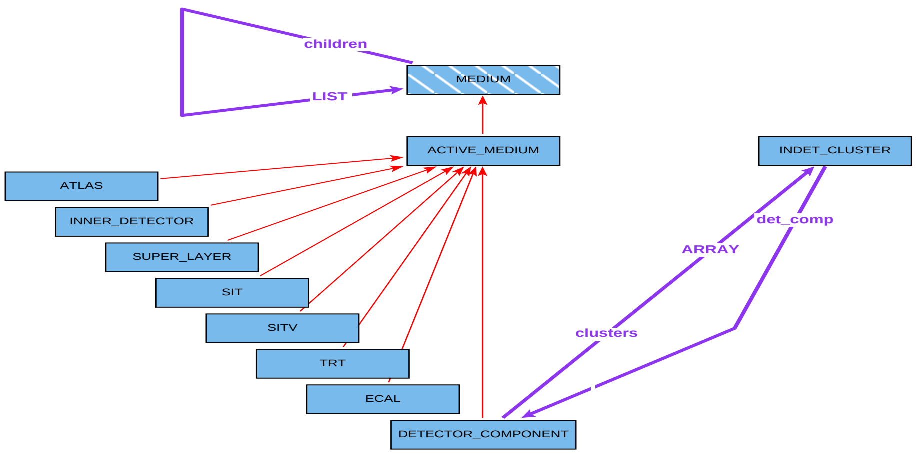

Figure 2: Media Class Diagram.

During track reconstruction, we are interested in only a small subset of the space points. We could just collect all clusters/space points of interest and then do track fitting, however we would like to preserve at least some of the structural support that the detector hierarchy provides. We have the requirement to "be able to partition clusters along a road, where each partition has at least one space poin"

So we have concluded that for each **ROAD** one can build a **TREE**(Fig. 3) with four levels, where each **NODE** is connected to a **MEDIA** or **CLUSTER**. This enables:

- navigating through and test whether each **NODE** is inside a **ROAD**,
- selecting and split off sub-trees of interest,
- having different reconstruction schemes working in parallel.

Object Persistency

We need random access to all objects at the time we use them (including all dependent objects). We also need objects to be updated on the disk at the right moment.

When fetching a persistant element it may need to be read in from disk or it may already be present. When an object is dragged in it brings with it all those

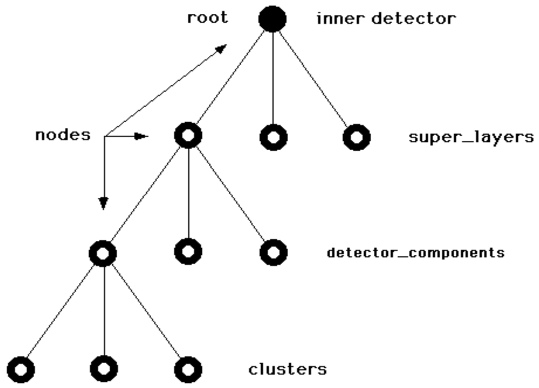

Figure 3: Tree.

objects directly referenced including persistent collections (but not the contents of these collections). This means that the user is not normally aware of reading data.

A persistent object must carry its OID. This is the only way an object can be identified on disk, an UPDATE flag indicating that it needs to be written out to disk, and information on which DBASE object it is connected to. On disk the OID is used to identify an object.

Calorimetry

The **ECAL** class corresponds to the ATLAS electromagnetic calorimeter, which has 3 samplings in depth.

The **ECAL** class provides clusters (**ECAL_CLUSTER**).

An **ECAL_CLUSTER** is an isolated region of energy deposition from one or more particles. It is characterized by three coordinates in space, for the energy-weighted centroid, and the its width in those coordinates.

An **ECAL_CLUSTER** is derived from the input calorimeter data which consists of groups of 5 words indicating non zero cell responses of the 3 layers of the calorimeter and the 2 calorimeter cell coordinates.

Tracking

- **CLUSTER** is basic measurement information for calorimeter and tracker, derived from digitisations (raw data). (A 'space point' is made from 1 or 2 clusters in tracker.)
- **SEED** is starting point to build road and begin track search.
- **ROAD**(Fig. 4) is envelope used for collecting tracker clusters in order to find tracks.
- **TRACK** is set of tracker clusters which fit electron hypothesis.

Tracking Schema can be desribed with the help of Class Diagram(Fig. 5) and Object Interaction Graph(Fig. 6).

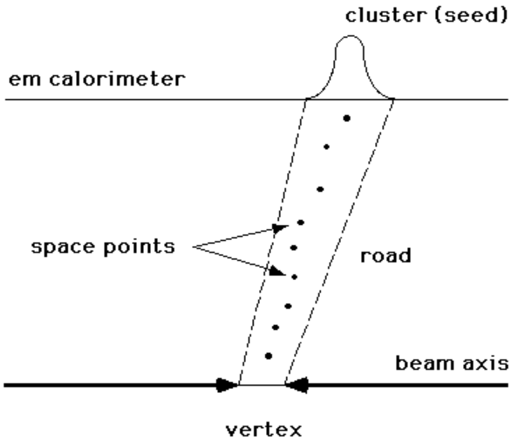

Figure 4: Tracking.

3.4 Algorithms

Algorithms, used mostly already in the current software, have been re-thought and re-written.

Calorimeter Clustering

The total energy deposited in the three layers of a square region in (η, ϕ), currently $3 * 3$ cells, is calculated. If this energy is greater than that in the four adjacent squares (moving ± 1 cell in η or ± 1 cell in ϕ), the centre of the square is taken as a candidate cluster. The square is moved one cell at a time in the eta direction, at fixed phi, and then the next row in phi is tested, until the centre of the square has visited every cell of the calorimeter. The energies in each layer of each cell of an $11 * 11$ matrix centred on each candidate are saved in a data bank.

Then the energy-weighted centroid in (r, ϕ, η) of each candidate is calculated, with the corresponding widths (and skew, but not correlations). This calculation is made for each of a set of rectangles, centred on the centre of the candidate, and the results saved in data banks. Currently, the set of $\eta * \phi$ rectangles is $3 * 3$, $3 * 5$, $3 * 7$, $5 * 5$, $5 * 7$, $7 * 7$.

Finally, the energy in each of a set of cones whose axis points to the (η,ϕ) cell at the centre of the candidate (not the centroid) is calculated and saved in a databank. These cones are circular, treating η and ϕ (the variables, not cell-numbers) as commensurable quantities, and currently have radii in $\eta - \phi$ space of 0.15, 0.20, 0.30, 0.40.

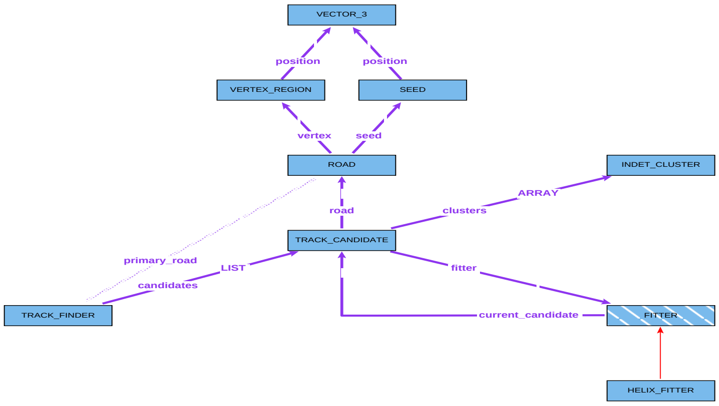

Figure 5: Fitter Class Diagram.

Helix Fit

The helix fit procedure[7] starts from these assumptions::

- Each **TRACK** will have about 6 **SPACE_POINT**s.
- A **ROAD** is used to collect **CLUSTER**s for a **TRACK**.
- Pattern recognition is done by searching for **SPACE_POINT**s in **DETECTOR_COMPONENT**s which have been partitioned along the length of a **ROAD**.
- **SPACE_POINT** is a precission measurement in 2 dimensions. A minimum set of **SUPERLAYER**s is needed to make a **SPACE_POINT**. If a superlayer has pixels : each pixel cluster becomes a **SPACE_POINT** because pixels are small very precise pats in two dimensions. However if a **SUPERLAYER** has strips : at least two strip **CLUSTER**s are needed to make a **SPACE_POINT** because a strip is short and precise in one dimension but long and thus very badly defined in the second
- **PARTITION** is minimum number of **SUPERLAYER**s needed to guarantee a **SPACE_POINT** (ignoring detector inefficiency, there should be 2 or more **SPACE_POINT**s per partition) The criteria to define are:
  1. the amount of overlap of a **SUPERLAER**s with a **ROAD**

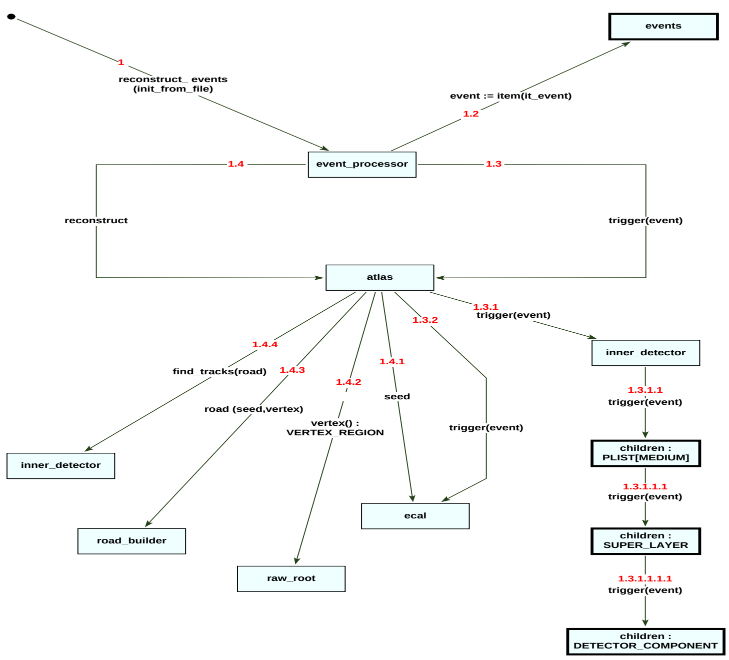

Figure 6: Event Loop Object Interaction Graph.

   2. the effieciency of each **SUPERLAER**

- cluster is the basic measurement infomation for calorimeter and tracker, derived from digitisations (**RAW_Data**). (A **SPACE_POINT** is made from 1 or 2 **CLUSTER**s in tracker.)
- **SEED** is starting point to build **ROAD** and begin **TRACK** search
- **ROAD** is envelop used for collecting tracker **CLUSTER**s in order to find **TRACK**s
- **TRACK** is set of tracker **CLUSTER**s which fit electron hypothesis.
- **EVENT** is raw and recontructed data associated with a single trigger

TRACK-fitting procedure can then be described in the sequence of steps:

1. Build a **ROAD** from **ECAL SEED** and **VERTEX**.
2. Collect a set of **SUPERLAYER**s and **CLUSTER**s which are in the **ROAD**.
3. Construct 3 **PARTITION**s: *start*, *middle*, and *end* in the following steps:
   - First, create 2 partitions:
     - a *vertex* **PARTITION** (from **VERTEX** outwards and
     - a *seed* **PARTITION** (from **SEED** inwards),

     where each **PARTITION** has about 1/3 of all **SUPERLAYER**s.
   - Next, select *vertex* or *seed* as the *start* **PARTITION**: Use the criteria of the smallest non-zero number of **SPACE_POINT**s. To allow for hard bremstrahlung, choose *vertex* for equal number of space points.

   - Next, build a *middle* **PARTITION** beginning from *start*.
   - Finally - the unselected *vertex* or *seed* **PARTITION** becomes the *end* partition, where the **SUPERLAYER**s which have been used in *middle* have been removed.
4. Begin a loop over **TRACK_SKELETON**s: Fit a **TRACK_SKELETON** using one **SPACE_POINT** from each of the *start* and *middle* **PARTITION**s and the **VERTEX**.
5. Use the skeleton **ROAD** to collect the **CLUSTER**s which are between the two **SPACE_POINT**s. Fit these **CLUSTER**s to make a **TRACK_SEGMENT**.
6. Collect all **SPACE_POINTS**s in the *end* **PARTITION** which are inside the current **TRACK_SEGMENT ROAD**.
7. Begin a loop over the *end* **PARTITION SPACE_POINT**s: For each **SPACE_POINT**, collect all **CLUSTER**s from the found set. and fit, keeping successful fits as **TRACK_CANDIDATE**s. End loops over *end* partition **SPACE_POINT**s and **TRACK_SKELETON**s.
8. Select tracks from **TRACK_CANDIDATE**s and provide a basic fit, a **VERTEX** fit and, if appropriate, a **SEED** fit.

In Track Fitting there are 3 steps:

1. A pre-check in which one assures that all of the **CLUSTER**s to be used in the fit are not already used for an other **TRACK_CANDIDATE**.
2. The trackfit it-self.
3. Quality checks of the fit on the
   - **VERTEX** compatability,
   - **SEED** compatability,
   - **MOMENTUM** spread,
   - chi squared, ...

Circular Pattern Recognition

The goal of this method is o find all **SPACE_POINT**s from the **ROAD** belonging to each **TRACK** (=circle). We use clustering in the space of the parameters of helixes (circles). After this procedure, **SPACE_POINT**s for each **TRACK** may be fitted.

3.5 Implementation

Data Flow

The Prototype reads

- the detector description (geometry)
- the data generated by the standard simulation program (Geant):
  - cell energies for the calorimeter
  - clusters for the inner detectors
  - results (tracks) found by Fortran Tracker

and writes output using ntuples, MOP and VRML.

Eiffel-Fortran Interplay

The Prototype can also be run together with the current ATLAS software by linking Fortran and Eiffel with a small number of C interface functions.

Calling can be done in both direction:

- **Eiffel calls Fortran** will provide encapsulation of the already existing (algorithmic) Fortran routines in the OO and use of Fortran libraries.
- **Fortran calls Eiffel** will be important for use of already developed OO code in Current software.

Using together Current software with our newly written code will enable us:

- Moving ATLAS to OO-structured software step by step by replacing parts of the Current software.
- Using different languages together (Eiffel,Fortran,C,C++).

We are aware of a risk of bringing Fortran-like constructs to the OO software.

Event Display

We are using **Virtual Reality Modelling Language** for the simple Detector and Event Display(Fig. 7).

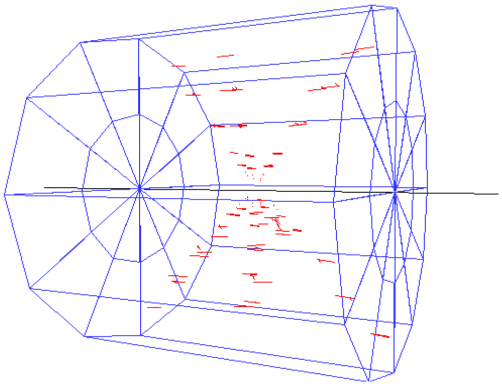

Figure 7: One event in the Event Display.

This approach has several advantages:

- Browsers are available.
- VRML is Object Oriented.
- VRML is simple.
- VRML contains Hyperlinks.

4 Status of the Project

After about one year of existence of the MOOSE Collaboration, first version of the **Prototype II is running**. The first Tracks have been already reconstructed. Our project started nearly from scratch (concerning the existing Eiffel code as well as developers skills in OO). Only about 10 people took part in the analysis and design phase of the project and about 5 were writing the code, most of them only part-time. These people were also distributed all over the Europe. Having in mind all these facts, the existence of the running prototype is a success of the OO technology. The general questions concerning MOOSE project are discussed in[2].

MOOSE Status Report will be presented to LCRB on November 7.

References

1. URL: http://www.cern.ch/OORD/Home_oord.html
2. C.Arnault: MOOSE: A Strategy for the Future, CHEP-95
3. V.Innocente: An Object Oriented Approach to CMS Reconstruction Software, CHEP-95
4. ATLAS Technical Proposal
5. URL: http://atlasinfo.cern.ch/Atlas/Welcome.html
6. S.M.Fisher,D.J.Candlin: Omo - A Tk/tcl based object modelling tool to support Eiffel, CHEP-94
7. A.Poppleton: IPATREC, ATLAS Software Note No.9