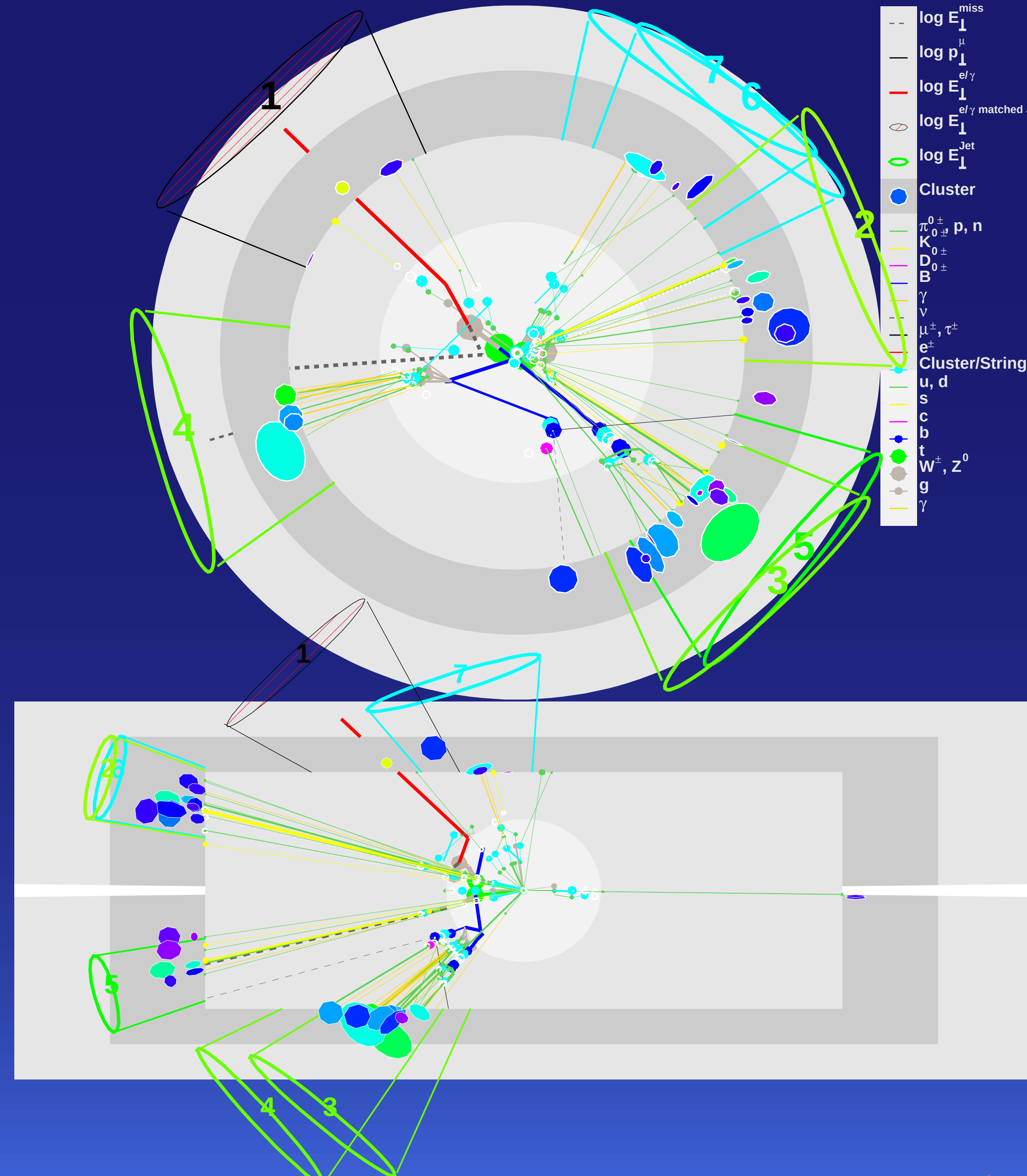
Physics Analysis Tools

MPP Atlas Meeting — Sven Menke, MPP München — 17. November 2008, MPP

- Introduction to PAT
- Tools for DPD making
- TAG files
- Analysis frameworks
- `AthenaROOTAccess` and interactive `athena`
- Tools for physics analysis
- Conclusions



- Physics Analysis Tools group collects and maintains code to perform physics analysis with `athena` and derived packages like `AthenaROOTAccess` in `python` and `C++`
- Input data format supported is `POOL/ROOT` files – i.e. high level objects as in ESD, AOD and DPD (primary & secondary)
- Also supported are TAGs (condensed information like the number of jets and electrons) which are either stored in a database or in `ROOT` files for the purpose of fast event pre-selection.
- The main categories for the tasks involved are:
  - to provide analysis skeletons to run in `athena` and `AthenaROOTAccess` on ESD/AOD/DPD level with and without TAG pre-selection.
  - to provide the code to create DPDs from ESDs and AODs and DPDs via skimming/thinning/slimming.
  - to provide the code to create TAGs from AODs.
  - to provide examples to add `UserData` in the DPD making step.
  - to collect (sometimes to maintain) general tools needed for physics analysis.
  - to provide frameworks on top of `athena` and `AthenaROOTAccess` to structure an analysis.
- Activities focus on recommendations from AMF Report `ATL-GEN-INT-2008-01`

Primary, secondary and tertiary DPDs

primary DPDs

- a subset of AOD objects after skimming, thinning and slimming from multiple AOD streams
- with the possible addition of analysis data (derived quantities like the mass of a composite particle candidate etc.)
- in the same format (`POOL/ROOT`) as AODs
- contents to be defined by the physics groups using them with the aim to share primary DPDs for many analyses (number of primary DPDs $\sim O(10)$)
- in some cases the primary DPD might be the AOD itself

secondary DPDs

- made from primary or prior secondary DPDs
- in the same format (`POOL/ROOT`) as AODs
- with more specific (i.e. smaller) content than primary DPDs but also likely more analysis data

tertiary DPDs

- made from primary or secondary DPDs in a different format (flat Ntuple)
- most specific selection cuts and analysis
- used for publication plots
- code and results need to be validate-able by reviewers
- no complex analysis should be done on tertiary DPDs

DPD production

- PAT needs to provide examples for primary DPD production as stated in the AMF report
  - standalone example to create a skimmed, thinned and slimmed DPD from AOD is available in `PhysicsAnalysis/DPDUtils/share/AODtoDPD.py`
  - for DPD making in the production system we have the JobTransform `Reconstruction/RecJobTransforms/scripts/fdr_makeDPD_trf.py` from Nicolas Kerschen et al.
  - slimming examples for truth, jets and tracks are in `PhysicsAnalysis/DPDUtils/share`
  - a python wrapper to use thinning directly from python (as in `PhysicsAnalysis/DPDUtils/share/semilep_ttbarFilterAlgorithm.py`) exists in `DPDUtils`
  - the examples provide AOD-based skimming – a TAG based skimming step can be attached prior to the DPD making
- physics groups need to define their own `AODtoDPD.py` fragments to customize
  - filter cuts and selection analysis
  - DPD output items (`StreamDPD`)
  - level of skimming, thinning, and slimming
- dedicated package `PrimaryDPDMaker` exists for this purpose with code from all major physics/performance groups

DPD production ► JobTransfroms for DPD making

Anastopoulos Christos, Davide Costanzo, Nicolas Kerschen

Usage:

- `fdr_makeDPD_trf.py [options] <inputaodfile> <outputaodfile> <maxevents> <skipevents> <jobconfig>`

Arguments:

1. `inputAODFile` (list) : Input file that contains AOD's
2. `outputAODFile` (str) : Output file that contains AOD's
3. `maxEvents` (int) : Maximum number of events to process
4. `skipEvents` (int) : Number of events to skip
5. `jobConfig` (list) : jobOption fragments containing the Gr'oup DPD settings

- Example jobO fragment to provide as 5th argument (`example.fdr_makeDPD_config.py`):

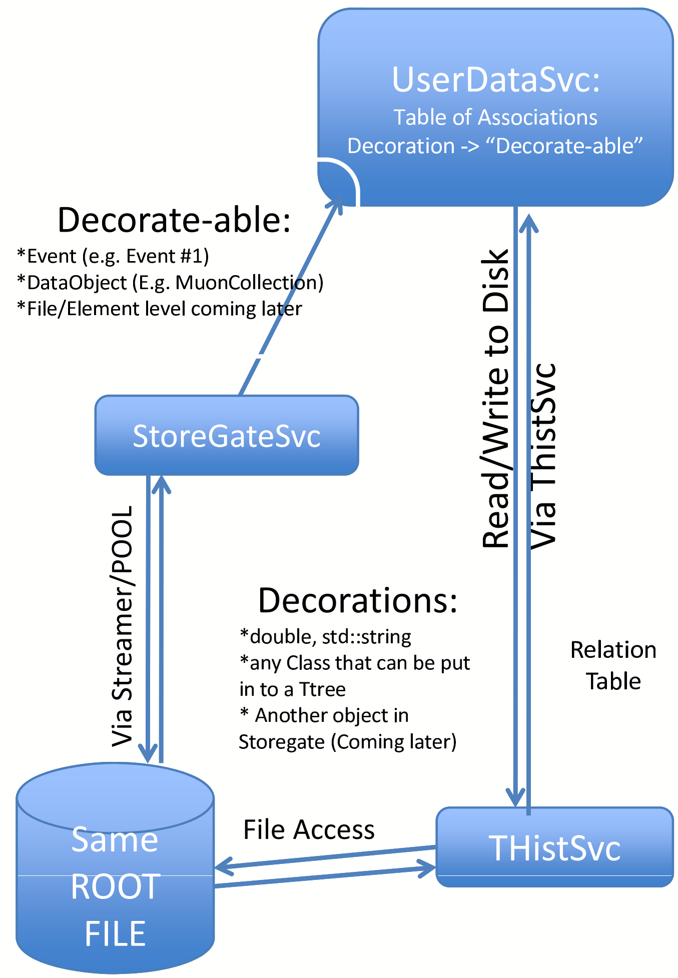
```
example.fdr_makeDPD_config.py        Tue Feb 26 10:15:14 2008        1
######################################################################
#
# DPD Group Configuration example
#
# Authors: Davide Costanzo, Nicolas Kerschen, Anastopoulos Christos
#=====================================================================


#Jet Slimming
include( "DPDUtils/slimJets.py" )
slimJets=slimJets("slimJets")
theApp.TopAlg += ["slimJets"]

#Track Slimming
include( "DPDUtils/slimTracks.py" )
slimTracks=slimTracks("slimTracks")
theApp.TopAlg += ["slimTracks"]

#Electron Filter Skimming
include( "DPDUtils/electronFilter.py" )
electronFilter=electronFilter("electronFilter")
theApp.TopAlg += ["electronFilter"]


#Truth Slimming
#include( "TruthSlimming.py" )

# Pool Converters for AOD
include( "LArAthenaPool/LArAthenaPool_joboptions.py" )
include( "CaloAthenaPool/CaloAthenaPool_joboptions.py" )
include( "TrkEventAthenaPool/TrkEventAthenaPool_joboptions.py" )
include( "RecAthenaPool/RecAthenaPool_joboptions.py" )
include( "ParticleEventAthenaPool/ParticleEventAthenaPool_joboptions.py" )
include( "G4SimAthenaPOOL/G4SimAthenaPOOL_joboptions.py" )
include ( "EventAthenaPool/EventAthenaPool_joboptions.py")
include ( "GeneratorObjectsAthenaPool/GeneratorObjectsAthenaPool_joboptions.py")
include( "InDetEventAthenaPool/InDetEventAthenaPool_joboptions.py" )

from AthenaPoolCnvSvc.WriteAthenaPool import AthenaPoolOutputStream
StreamDPD = AthenaPoolOutputStream( "StreamDPD" )

######################Write Item List######################################
StreamDPD.ItemList =  ['EventInfo#*', 'TrackRecordCollection#*']
StreamDPD.ItemList += ['ElectronContainer#ElectronAODCollection']
StreamDPD.ItemList += ['egDetailContainer#egDetailAOD']
StreamDPD.ItemList += ['Analysis::MuonContainer#StacoMuonCollection']
StreamDPD.ItemList += ['ParticleJetContainer#Cone4H1TowerParticleJets']
StreamDPD.ItemList += ['ParticleJetContainer#Cone4TruthParticleJets']
StreamDPD.ItemList += ['Rec::TrackParticleContainer#TrackParticleCandidate']
StreamDPD.ItemList += ['Rec::TrackParticleContainer#StacoTrackParticles']
StreamDPD.ItemList += ['VxContainer#*']
#StreamDPD.ItemList += ['TruthParticleContainer#SpclMC']
# Use this if thinning GEN_AOD
# StreamDPD.ItemList += ['McEventCollection#GEN_DPD']
# StreamDPD.ItemList += ['TruthEtIsolationsContainer#TruthEtIsol_GEN_DPD']
StreamDPD.ForceRead=TRUE;  #force read of output data objs

#electron Filter
StreamDPD.AcceptAlgs=["electronFilter"]


#==============================================================
#
# End of job options file
#
###############################################################
```

Sebastien Binet, Paolo Calafiura, Charles Leggett, Yushu Yao et al.

- Aim is to provide a common `UserData` solution outside `EventView`
  - persistifiable on `POOL/ROOT` files (i.e. ESD/AOD/DPD)
  - to store intermediate analysis results (like fit results, combined particle masses, event shapes etc.)
  - likely some sort of value (`int, float, double`) or `vector` of these and a label
- Current status:
  - `UserDataSvc` is in the release since `14.1.0`
  - entire events or SG objects can be ”decorated” with the `UserDataSvc`
  - The decoration consists of a label (e.g. `HiggsMass`) and an object (virtually anything that can be put into a `TTree`) – in this case a simple `float` would do ...
  - The decorated object can be associated with other SG objects
  - decoration info (label, actual decoration object, associations) is written into a `TTree` in the same file the `CollectionTree` ends up in
  - reading the file back in with athena or ARA allows access to the decorations

UserDataSvc – Adding User Analysis Data

- **Use UserDataSvc to add/read user analysis data.** E.g. (only to show the possible use cases):
  - Event #243 looks like a Higgs Event and the reconstructed Higgs Mass is 118 GeV.
  - I'd like to keep a record of how many Muons hit my detector section.
- **What Can Be Decorated ("decorate-able"):**
  - Event (e.g., Event #1, #2, etc)
  - DataObject (e.g. MuonCollection)
- **Types of Decorations:**
  - Any Class that can be put into a ROOT tree.
  - Predefined double and std::string for easy usage.
- **Mechanism:**
  - UserDataSvc keeps an association table between decorations and "decorate-able"
  - UserDataSvc utilizes THistSvc the same ROOT file that StoreGate is reading/writing.
  - Decorations are saved in the ROOT file as branches of a TTree. Allowing TTree.Draw() of Decorations in AthenaROOTAccess.
- **Preliminary Functionality and Examples will be available in 14.1.0**



Contributors: Paolo Calafiura, Sebastien Binet, Charles Leggett.
Code Implementation: Yushu Yao

8-Apr-08

17

TAG files

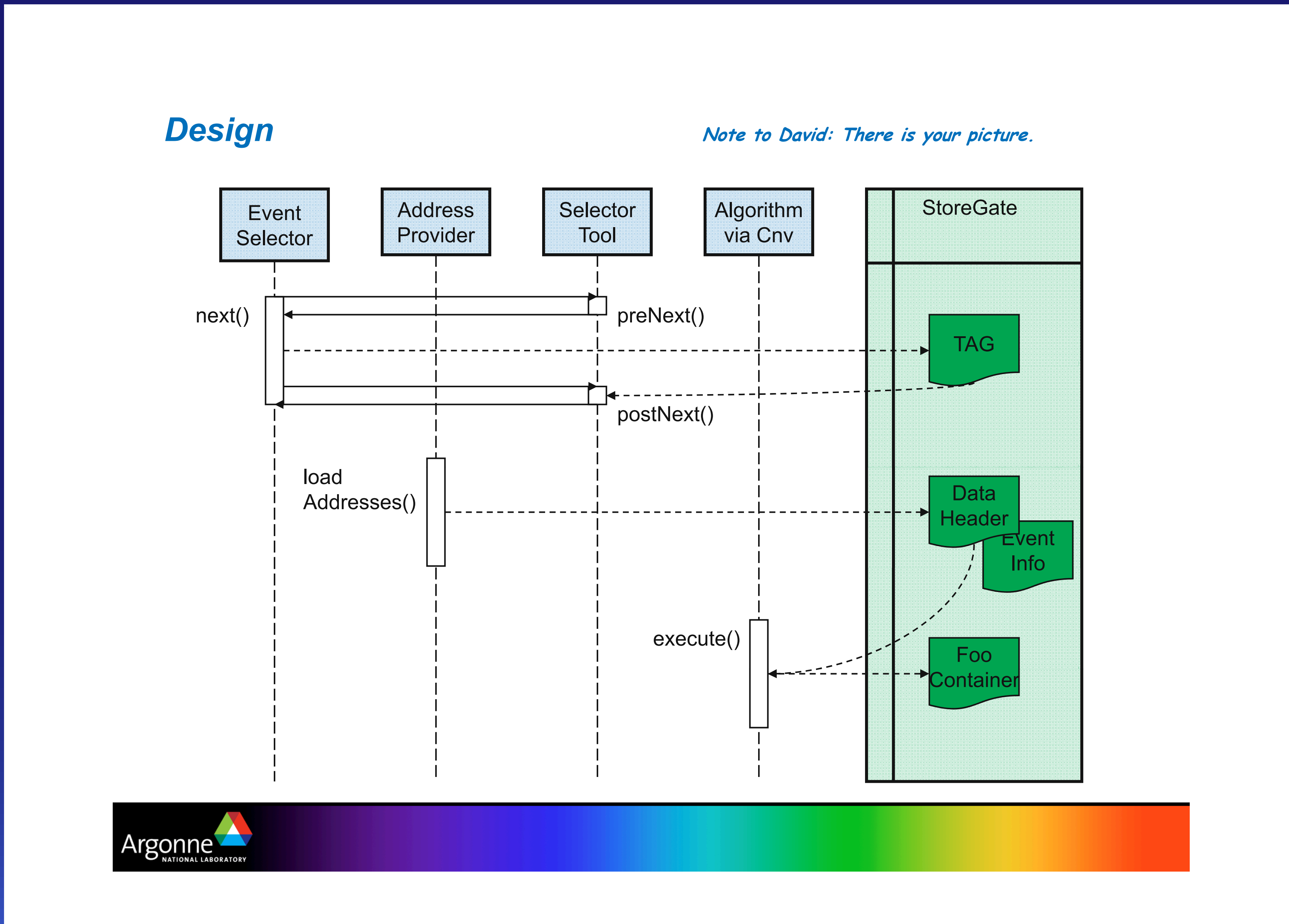
David Malon, Peter van Gemmeren et al.

- TAG attributes are a very efficient way of characterizing an event
  `https://twiki.cern.ch/twiki/bin/view/AtlasProtected/TagForEventSelection14`
- SQL database and flat `ROOT` files contain TAG info
- same `athena` query syntax for both options
- Event Level Quantities
  - Run Number/Event Number
  - Event Type
  - Number of Tracks
  - Primary Vertex
  - Lumi Block
  - Missing Et, ϕ, SumEt
  - ...
  - Data Quality Flags per Subdetector
  - Trigger Information CTP decisions, Lvl1 type, Lvl2/EF masks

TAG files, continued ...

David Malon, Peter van Gemmeren et al.

- Object Level Quantities
  - Electrons/Photons/Muons/Taus/Jets
  - falling pT ordered
  - e/γ/μ: loose pT, ϕ, η, tightness
  - tau/jet: pT, ϕ, η, likelihood (tau/B-jet)
  - ...
- Physics TAG
  - one for each phys/perf group to be defined by them
- `athena` has been extended to allow computational processing of TAG attributes without reading the event data (not even the header)
  - can check proximity of 4-vectors not only number of objects above certain pT
    - try finding the closest jet to an electron in SQL instead ...


Design
Note to David: There is your picture.
Event Selector
Address Provider
Selector Tool
Algorithm via Cnv
StoreGate
next()
preNext()
TAG
postNext()
load Addresses()
Data Header
Event Info
execute()
Foo Container
Argonne
NATIONAL LABORATORY

Analysis frameworks

- Several analysis frameworks exist on top of `athena` and/or `AthenaROOTAccess`
- `EventView` Kyle Cranmer, P.A. Delsart, Amir Farbin, Peter Sherwood, Akria Shibata et al.
  - runs in `athena`
  - inserts objects, identifies overlaps, combines objects and analyzes them
  - can be persistified in `POOL/ROOT` files
  - became more modular since the time of the AMF
- `EWPA` (Every Where Physics Analysis) Massimilano Bellomo et al.
  - a newer lightweight alternative to `EventView`
  - runs in `athena` and `AthenaROOTAccess`
  - can also be persistified on `POOL/ROOT` files
  - both `EV` and `EWPA` can use common tools and dump DPDs or flat ntuples
- `AMA` (Atlas Modular Analysis) Max Baak, Giuseppe Salamanna et al.
  - runs analysis on ESD/AOD/DPD in `athena` and `AthenaROOTAccess` and creates flat ntuples/histograms
  - is modular and uses internal EDM to access simple types for data

Analysis frameworks, continued ...

- several `python` based skeletons
  - mostly to structure output and input files
  - run standard `athena` or `AthenaROOTAccess` otherwise
  - use the `athena` EDM
  - can use common `athena` tools
  - write DPDs or flat ntuples
  - are the prefered choice for PAT
  - look at PAT wiki for recent tutorials
    e.g. `https://twiki.cern.ch/twiki/bin/view/AtlasProtected/DPDMakingTutorial140220`
  - good example following this design is the `AthenaROOTAccess` based analysis by our top-mass group in CVS: `groups/MPP/ARA_Examples_Top`

What is AthenaROOTAccess?

https://twiki.cern.ch/twiki/bin/view/AtlasProtected/AthenaROOTAccess by Scott Snyder et al.

- AthenaROOTAccess allows you to access the objects in ESD/AOD/DPD directly from ROOT without the athena framework
- Many athena classes (most notably the classes describing the transient objects) are available from ROOT and PyROOT via their dictionaries
- The athena software has to be installed and setup but instead of athena.py you run python -i or root

How does it work?

- Minimal test.py start script for one AOD:

```
import user
import ROOT
import PyCintex
import AthenaROOTAccess.transientTree
f = ROOT.TFile.Open ('AOD.pool.root')
tt = AthenaROOTAccess.transientTree.makeTree(f)
```

- sets up a virtual transient tree with branches corresponding to the transient objects identified by their StoreGate keys
- The transient/persistent converters (TPCnv) are automatically invoked when a specific entry is requested to convert from the persistent data on the ESD/AOD/DPD to the transient representation
- for example the branch CaloClusterContainer_p4_CaloCalTopoCluster in the persistent CollectionTree on the AOD will trigger the creation of the branch CaloCalTopoCluster in the transient tree which points to the transient CaloClusterContainer

- Minimal `chain.py` start script for many AODs:

```
import user
import ROOT
import PyCintex
import AthenaROOTAccess.transientTree
CollectionTree = ROOT.AthenaROOTAccess.TChainROOTAccess('CollectionTree')
CollectionTree.Add('AOD.*.pool.root')
tt = AthenaROOTAccess.transientTree.makeTree(CollectionTree)
```

Examples

- `python -i chain.py`

```
tt.Draw('CaloCalTopoCluster.e()')
ce = ROOT.ClusterExample()
ce.plot(tt)

cc = tt.CaloCalTopoCluster

tt.GetEntry(0)
cc.size()
c = cc.at(0)
c.e()
```

- `root`

```
TPython::Exec("execfile('chain.py')");
CollectionTree_trans->Draw("CaloCalTopoCluster.e()");
ClusterExample ce;
ce.plot(CollectionTree_trans);
TBranch * cbr = CollectionTree_trans->
                GetBranch("CaloCalTopoCluster");
const CaloClusterContainer * cc =
   *((CaloClusterContainer **)cbr->GetAddress());
cbr->GetEntry(0);
cc->size();
CaloCluster * c = cc->at(0);
c->e();
```

Typical development cycle

- Either use python or compiled C++ to develop your analysis
- Use CINT only to execute python script and instantiate compiled C++ classes
- in python no compilation needed but typically 2 times slower than compiled C++ code
- Compiled C++ code needs to come with a dictionary to be visible from CINT and python
  - ► typically as easy as adding the class to the selection.xml and XXXDict.h files in the package
  - ► lookup examples in PhysicsAnalysis/AthenaROOTAccessExamples
- To get started check out PhysicsAnalysis/AthenaROOTAccessExamples and add a class ...
- Use TBrowser to see all the EDM classes available on the current TTree from ESD/AOD/DPD
- Going back and forth between athena and AthenaROOTAccess is easy since only the retrieval of the containers differs (StoreGate vs. TTree)
  - ► can develop analysis in AthenaROOTAccess and port with almost no effort to athena later

Limits

- need special dual use tools to run in both athena and AthenaROOTAccess
- databases, detector description, identifiers, services are not available in AthenaROOTAccess
  - ► code that needs these has to reside in athena and make output available on DPD (maybe as UserData) which can be analyzed in AthenaROOTAccess
  - ► note that one can combine athena and AthenaROOTAccess to get access to Identifiers and other (slow/not changing) conditions data/geometry; see for example recent Artemis tutorial: https://twiki.cern.ch/twiki/bin/view/AtlasProtected/CaloRecTutorial140220

Yushu Yao et al.

- AthenaROOTAccess wrapper
  - re-using code inside athena and in ARA needs some wrapping machinery since most Gaudi based classes like AlgTools, Services etc. do not work in ARA
  - code from from Yushu Yao et al. is in the release since 14.1.0
  - wiki page linked from https://twiki.cern.ch/twiki/bin/view/Atlas/AthenaROOTAccess page:
    https://twiki.cern.ch/twiki/bin/view/Atlas/ARAAthenaDualUseTool
  - mainly to get an AlgTool on the athena side and a direct callable tool in ARA
  - with the possibility to set properties in ARA
  - AraTool/ARAToolBase is the base class for this
  - your AlgTool needs to derive from this base class
  - look at examples in AraToolExamples

PyAthena and python based thinning

Sebastien Binet et al.

- New PyAthena framework in Control/AthenaPython improves the way python algorithms are treated in athena
  - like the C++ algorithms the configuration step and creation step are separate now for python algorithms
  - this is important to be fully Configurables compliant, increases speed and debugging efficiency
- Documentation and Tutorials
  - extensive wiki page: https://twiki.cern.ch/twiki/bin/view/Atlas/PyAthena
  - Please try it and provide feedback to Sebastien!
- Thinning Example
  - Now (rel14) fully in python (no more wrapper needed)
  - look at wiki for actual example code
- Utilities provided by PyAthena
  - class lookup in PyAthena like PyCintex or ROOT
  - retrieve services and tools via PyAthena.py_svc and PyAthena.py_tool
- StoreGate improvements
  - py_retrieve now takes only 50% slower than C++ retrieve from StoreGate
  - py_record about 100% slower
  - both bound by Reflex/PyRoot overhead
  - now both fast enough for full analysis in python!

Tools for physics analysis

- Recently PAT re-focussed efforts in the direction of common analysis tools and less in the direction of frameworks
  - common framework independtent tools make the choice of a particular framework less important
  - common tools can be validated more easily and provide confidence in the results obtained by them.
  - in several areas critical tools are either missing or not known to PAT.
  - to make PAT useful as entry portal for analysis an almost complete list of relevant tools needs to be present on the PAT web sites.
- Tools currently on the wishlist are:
- Overlap Tools
  - checking for overlap, make a selection, re-reconstruct jets, MET, etc. framework independent and consistent
- Trigger Tools
  - Trigger object to analysis object association (and vice-versa)
  - Trigger analysis in `athena/AthenaROOTAccess` (Till Eifert, Joerg Stelzer)
- Isolation tools for different type of objects
  - tracks, clusters, jets, combinations based on actual constituents and not on simple ΔR
  - electron isolation tool (Inga Ludwig, Jochen Hartert, Ralf Bernhard)

Tools for physics analysis, continued ...

Event subtraction/replacement tools

- AOD based 0.1×0.1 grid of `CaloTowers` with 2 layers (EM and HAD) to cross-check noise, PileUp, UE, Isolation, or even replace MC by data and vice-versa

Metadata Tools

- Bookkeeping for selection efficiency (David Cote)
- Type of Simulation Metadata (w/o Overlay, Pileup, etc.)
- Browsing of (sub-detector) data quality flags and selection tools based on this (Max Baak)

Efficiency Measurement and Bookkeeping Tools

- for efficiency measurements, monitoring and corrections (Arno Straessner, Matthias Schott)

MC correction tools at AOD level

- AOD to AOD correction mechanisms (for MC) (Mark Hohlfeld, Alfio Lazzaro)
- Corrections to be run on the fly during an analysis job (on MC)

Data correction tools at AOD level

- for re-calibration with improved calibration constants on AOD level

Data Handling tools

- FileStager – Tools for doing local analysis on nearby (same building) grid collections (Max Baak)

Conclusions

- Analysis Model
  - based on AMF report ATL-GEN-INT-2008-01
  - DPD as POOL/ROOT file well established
  - analysis based on C++ and python supported
  - first athena to create DPD then AthenaROOTAccess to analyse DPD
  - TAGs made more useful by adding computing step
- Frameworks
  - follow the simpler is better approach
  - frameworks need to be modular, use common tools and be validated
  - analysis should be done on AOD/DPD (not flat ntuples)
- Tools
  - recent effort to document all useful analysis tools on PAT pages
  - several missing but cruicial tools identified
  - please look at the PAT wiki pages, join the PAT phone meetings (every other Wednesday 16:00-18:00; 19th November the next) and provide feedback