

Physics Analysis Tools Workshop Summary Report

F. Akesson, K. A. Assamagan*, D. Best, S. Binet,
P. Calafiura, C. Collins-Tooth, K. Cranmer, J. Cranshaw,
S. Dean, F. Gianotti, N. Konstantidinis, W. Lavrijsen,
C. Leggett, P. Loch, T. Maeno, D. Malon,
D. R. Quarrie, S. Rajagopalan, P. Sherwood, S. Snyder

Abstract

The current status of various aspects of physics analysis tools are reviewed in the general context of the software development for analysis. Feedback from the user community and user requirements on the tools for physics analysis are necessary in shaping the improvement of the existing tools and the development of new ones. Solutions to known problems are presented and discussed.

1 Introduction

The first workshop of this series, since the physics analysis tools group was formed, took place at the University College London (UCL) on April 5-7, 2004. At the time, there were many different frameworks for analysis and the data formats known as Event Summary Data (ESD) and Analysis Object Data (AOD) were not defined in term of their contents and structures. The objective of that workshop was to study the different approaches to the analysis domain, to identify commonalities and to propose a baseline, unified framework for analysis. A summary of the UCL workshop can be found here [1]. Following the UCL workshop, the first implementation of the AOD classes (integrating the on-going work of the AOD/ESD Definition Task Force [2]) and AOD building tools were completed and the first tutorial on doing analysis on ESD and AOD took place on June 21 2004 at CERN. The implementation of event tags for the selection and collection of interesting events soon followed in collaboration with the database group. These event tag tools have been used in the Tier 0 exercise. A second tutorial was organized during the physics workshop in November 2004 at CERN. As the interactive analysis in ATHENA [3] was becoming widely used, there

*Corresponding author: ketevi@bnl.gov

have been many requests from users leading to the improvement and development of other tools. A third tutorial on using ESD and AOD in analyses was organized during the North American Physics Workshop in December 2004. At the time of this writing, ESD and AOD are being produced from the samples generated for the up-coming physics workshop in Rome. To help the growing user community of ESD and AOD, a fourth tutorial was organized during the software week of February 2005.

The analysis framework proposed during the UCL workshop has taken a foothold and looks viable:

- More and more people are using the ESD, AOD and the tools developed for them in their analyses.
- Many problems reported by users are fixed by the developers.
- Various requests from users are implemented and integrated.
- Some users have taken a critical look at the current framework of analysis on ESD/AOD/Tag and have given constructive criticism.

At the time of the Tucson workshop on physics analysis tools, the following analysis options are available to the user community:

- The creation of combined NTuples (CBNT) from the reconstruction and the use of ROOT [4] macros to operate on the NTuples. This has been around long before the UCL workshop.
- The creation of ESD, AOD (and eventually event tags and event collections) from the combined reconstruction leading to analysis codes in C++ as ATHENA algorithms or analysis codes as Python [5] algorithms or scripts.
- Interactive analysis in ATHENA.

Access to ESD and AOD directly in ROOT is not a viable analysis option at the moment. This is due to a number of technical complications that may be resolved in the mid-term.

Given the current state of the analysis domain, as thus far summarized, the following topics were discussed during the Tucson workshop:

1. Integration of currently known user requests and requirements. Feedback from the user community is recognized as important in this step.

2. The data format, if any, to use beyond the AOD: processing large statistics on AOD is reported as being slow and transforming the AOD into NTuples is not recommended since important tools such as back navigation (the links to the ESD and the to raw data) are no longer applicable. The *event view*, discussed in great detail during this workshop (see later in this document), may offer the solution of "NTuplelizing" the AOD while keeping needed features of POOL (Pool Of persistent Objects for LHC) [6], thus maintaining the back link to the AOD and ESD within the ATHENA framework.
3. The evolution of the interactive analysis tools and their integration with the event display and visualization tools.
4. The tools for removing redundancies, overlaps and ambiguities in the AOD. Since some of these may be physics analysis dependent, a set of tools are proposed as examples/guidance to the user.
5. ROOT versus AIDA histogramming tools.
6. Issues (and proposed solutions) regarding object navigation and object associations. These issues are closely connected to StoreGate's SymLink.
7. Evolution of fast simulation tools.

2 ATLAS Project Based Build

The basic strategy for the ATLAS project based build has been defined [7] with a baseline set of projects which consists of core, detector description, event, simulation, reconstruction and analysis. It is not obvious in which project some packages will go, e.g., packages needed by the on-line monitoring or for calibration. The final project decomposition might evolve. The tag collector II support for the project based build is in place with the auto-tagging of containers. The support for hierarchical auto-tagging of project packages will go into production soon after this workshop. The prototype AtlasCore project is in place, based on the release 10.0.0, except for the missing dependencies on Nova (to be dropped by the release 10.2.0). Most of the other projects are currently underway. The AtlasCore project prototype is expected to be completed by the end of April; all the project prototypes will be available by the end of June 2005, to go into production by mid-September, although there might be many iterations once the prototypes are available. The distribution kits will need to be rethought in the context of the project based build: today, Pacman retrieves

a package and all the other packages upon which the needed package depends. It is not yet clear how this will work across different projects.

3 LCG Application Area Reorganization and its Implications

The reorganization of the LCG application area includes the merging of SEAL (Shared Environment for Applications at the LHC) [8] into ROOT, the merging of the POOL storage manager (otherwise unchanged) into ROOT. SPI and simulation projects are unaffected. There seem to be two schools of thought on the implications of the merging: on the one hand, it is seen as an optimized merge of the two cultures and client bases, leading to the reduction of duplications and improved libraries. On the other hand, the merging is seen as the absorption of SEAL into ROOT, improving ROOT but keeping its culture, naming and packaging convention. ATLAS, CMS and LHCb would like to see an internal reorganization and repackaging within an extended ROOT, leading for example, to light weight vector and linear algebra libraries, a plug-in manager not based on CINT. Further, the experiments also express the need to see a detailed plan of this merging/repackaging quickly. The overall proposal for combining CINT/Reflex [9] dictionaries looks good, to be discussed further during the dictionary workshop in May 2005. PyReflex and PyROOT [10] will merge: technically they cannot merge, thus this will be the “killing” of PyReflex but we will have one tool with the capabilities of both. New Math libraries are proposed, to incorporate the best of both the SEAL and the ROOT ones. We need to understand the packaging and the naming convention in these math libraries. Furthermore, it is not clear how the new math libraries relate to CLHEP [11]. The migration of the ATLAS software to CHLEP 1.9 is on hold for a month or so in order to understand the issues better; the Geant4 [12] team does not want to migrate from CLHEP.

The experiments can still influence the direction taken by the new ROOT project as the CERN organization does put the experiments into the management hierarchy. However, there needs to be a quick “litmus test” to see whether the new organization will be responsive and flexible. In this respect, there is the proposal that Physics Vector and Linear Algebra Libraries be used as such a test; but this implies the migration from CLHEP : it is not clear if this is desirable.

4 Interactive Analysis

There are two models for the interactive analysis. In the first model, the analysis is done in ATHENA. One uses Python as a binding to ATHENA and ROOT, and also for rapid prototyping. This is the baseline model. From initial feedback, users seem to accept it. The second model is the PAW/ROOT-style analysis, that is, CINT prompt instead of Python prompt and ROOT CINT is used to access ATHENA from ROOT. There is clearly an interest in the community for this ROOT based alternative to the interactive analysis. It is possible to satisfy both communities when the two dictionaries, Reflex and CINT, are unified — a unification that will allow for full interoperability between CINT and Python. In the near term, there is effort to make the event view objects accessible from ROOT and ATHENA, which should satisfy the most immediate need for interactive analysis.

With the introduction of Python, the user has to learn another language. What does Python buy us that C++ cannot provide? It is not necessarily the number of coding statements (C++ versus Python) but also the ease of use. Python lets one concentrate on the problem to solve while in C++, one may have to worry about the C++ apparatus and the overhead. Python has a large library of standard modules available that can be readily imported, thus simplifying some tasks. Many non-Python libraries come with Python bindings and are thus also easily available in Python. Python is good for prototyping. But for batch processing of large statistics, C++ should be used. If one is fluent in both languages, there is no problem. There is a worry about the algorithmic part of Python: if one wants to write a complicated algorithm such as a clustering algorithm, one can write the algorithmic code in C++ and use Python as glue (just create the dictionary for C++ algorithm) to make the algorithm available in Python. But the reverse is not so straightforward: complicated algorithms written in Python are not easily accessible in the C++ side: this is worrisome. It was therefore proposed that any algorithm/tool should be made available as C++ API even if the internal development or coding is in Python. It should be packaged to be usable globally, i.e., in the C++ side too. It is the responsibility of the person who writes the algorithm/tool in the Python to provide the C++ interface.

We need to demonstrate that this Python interactive environment is viable by getting more people involved. More complex analyses should also be exercised. Beginning examples of complete analysis algorithms in Python should be made available in the CVS repository under PhysicsAnalysis soon. Examples of using Python as a glue to C++ are already available.

The interactive access to MC truth data in the AOD requires the implementation of an iterator protocol, thus the introduction of the PyKernel/PyIter classes. Currently there is a small problem due to differences in the implementation of the iterator in C++ and Python – this should be fixed in the near future.

Ideally, the interactive analysis environment should allow for the following:

- Ability to seek to, skip, and reprocess an event.
- Access to all levels of the data hierarchy including conditions data
- Access to event generation and simulation
- Execution of the reconstruction algorithms on raw data or on the ESD
- Analysis at the AOD, NTuples or Event View levels
- Data, algorithm, task browsing, and the event display and visualization (see the section on the event display for details)
- Transparent transition from local to GRID analysis (this has not been demonstrated yet).
- Access two event collections (*e.g.* signal and background) simultaneously.
- The possibility of modifying, recompiling and executing code without exiting the interactive prompt. This is not possible for C++ libraries since outstanding references cannot be located. But for Python modules this can be achieved; indeed, the PyBus tool (part of SEAL) can pervasively change all references to anything in one module by the equivalent references in another module.

One needed feature of the interactive analysis is the ability to seek to and read in an arbitrary event from the input data stream; in general, the ability to re-initialize the event loop without exiting the interactive session. A prototype tool to do this exists, PyPoolSeek, but it does not call the `reinitialize()` of the event loop manager; this should be done in the next version of Gaudi. In the meantime, the code is available and usable with customized versions of some classes. The event counter used in the PyPoolSeek tool is sequential. The extension of this tool for the run number in addition to the event number, i.e., `seek(runNumber, EventNumber)`, and for suppressing the processing of intermediate events when one needs to get to a specific event would be good to have and should be investigated; It was recommended to initiate discussions with the relevant developers so as to get the customized versions

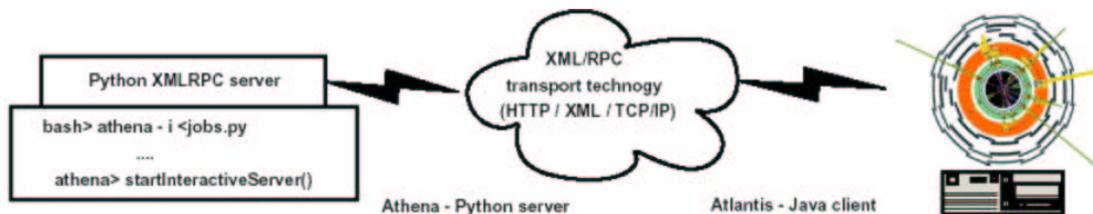


Figure 1: A schematic representation of the ATHENA - XMLRPC - Atlantis system used to produce an interactive Event Display.

of the classes needed for this tool in the release and to provide the API for the `reinitialize()` method of the event loop manager in the next version of Gaudi.

Two other features are needed to make the interactive analysis truly useful. The processing speed, i.e., to be able to run over a moderately-sized sample and make plots in few seconds. At the moment, we are at the level of few minutes; a caching schema should be considered to improve the processing time. The other needed feature is the ability to read different multiple samples in a single interactive session, e.g., open both signal and background samples, make histograms on each and overlay the histograms.

5 Interactive Analysis and Event Display

Atlantis is moving in the direction so that it will be able to do everything available in interactive ATHENA. The development surrounds three related technologies: interactive analysis in ATHENA, the XML RPC Server, and Atlantis. It was demonstrated that:

1. the `EventDataToXML` algorithm used to make the input files to Atlantis can be run interactively,
2. the resulting XML file can be transmitted via XML RPC,
3. and Atlantis (running as an independent process) can listen to the RPC port and update its display.

This is shown schematically in Figure 1. Furthermore, it was demonstrated that a user could change a property of the `EventDataToXML` algorithm (for instance changing from cone jets to Kt jets) and update the event display.

It was also shown that one can run a remote interactive ATHENA session and steer it with XML RPC. In that setup, one asks the server to execute an interactive

ATHENA command on a remote interactive ATHENA session. Essentially, this supplies the reverse form of the communication. The plan for Atlantis is to be able to transmit back information to the interactive prompt.

Since the release 10.0.0, Atlantis is an external package, i.e., one gets automatically Atlantis and Atlantis_new as aliases after the ATHENA setup and it is copied locally with the software distribution. Several new features, such as missing energy, color coding for the calorimeters, etc, will be included by the time of the tutorial on Atlantis during the software week in May 2005 (some of these new features are already in Atlantis_new). For the display of AOD objects, the list of the AOD and their connections to the ESD are needed, to be provided by the physics analysis tools group.

It was also discussed if Atlantis could rely on access to StoreGate directly. This may be considered in the future as the use case for this becomes more clear. It would be difficult to merge with other event display packages. Indeed, at the moment, there is no plan to merge as Atlantis is in Java and the other event display in C++. It was proposed that the ATLAS community tests both event display tools, then decides whether merging both would be the course of action to take. One needs to understand how to get the best aspects of both: it may be difficult to merge but at least one can access each as an independent application.

6 SymLink

SymLink is a tool which allows one to record a container of objects as one type and retrieve it as a container of a different type. For example, through SymLink, one can record a container of electrons and later, for whatever reason or purpose, retrieve the same container as a container of IParticles — the Electron and the Muon classes derive from the IParticle class but the ElectronContainer or the MuonContainer classes do not derive from the IParticleContainer class. How to relate the ElectronContainer to the IParticleContainer is known as SymLink. Many analysis use cases of SymLink exist, for example, when one would like to treat the container of Electrons and Muons as containers of IParticles with no regard to the detailed differences in the Electron and the Muon implementations. Although the current implementation of the SymLink works for most use cases, it does not work in certain circumstances and it is not compiler safe nor portable. In order to avoid the potential problems inherent to the SymLink implementation, prior to the workshop, the SUSY group has implemented a solution where a copy of the containers are made with the new container not owning

the contained elements — this is known as a view container. However, there are potential persistency problems with the view container solution: the view vectors must be persisted with original AOD for this to work. An alternative solution for the view container would be to use `ElementLinkVector`, but this breaks compatibility with the container type.

The discussion during the workshop centered on finding a compiler safe, portable, trustworthy and durable implementation of the `SymLink` tool. The proposal at the workshop is to relate the containers (for example `ElectronContainer` and `IParticleContainer`) by an inheritance structure in the same way as in the contained elements (the `Electron` derives from the `IParticle`). However, for this solution to be viable, two issues should be addressed. First, the type-checking of pointers on insertion: one must ensure that illegal operations are not allowed, such as attempting to insert an `IParticle` into an `IParticleContainer` `SymLinked` to an `ElectronContainer` (where the `IParticle` is not an `Electron`). This can be done dynamically at runtime or statically with an additional class, the `ConstDataVector` which would be used in the inheritance hierarchy and as a base class of the `DataVector` class. The consensus was to do the type-checking of pointers dynamically at runtime. Second, the pointer conversion on extraction. Three possibilities were considered:

1. dynamic casting at extraction,
2. the `ConstDataVector` remembers the pointer conversion offset from the base type,
3. or the `ConstDataVector` at the top of the inheritance hierarchy holds a table lookup of pointer conversion offsets.

It was proposed to do an initial implementation of the `SymLink` with the dynamic casting at extraction.

7 Tag, Streams, Physics Analysis : Near Term Planning

The initial AOD created at the Tier 0 is a disjoint partition of events in the sense that each event is written exactly once and similar events are written to the same file for storage optimization. The event collections are pointers to events in persistent storage, along with event-level meta data (the tags) used for event selection. The collections are arbitrary: they may span many streams, and a given event may appear in

different collections. Physics groups and users may extract copies of their interesting events by querying the tag database and filling their dedicated AOD samples. They can also build new collections (probably ROOT-based) containing only the results of the selection; or, possibly in the future, the selection criteria will be recorded in the data management system. The usefulness and some feasibility tests of the current tag-based infrastructure are being carried out as an input to the computing technical design report. It was also proposed (before the workshop) to build collections for the Rome Initial layout data at two levels: a master tag collection for all the Rome data in MySQL or Oracle, resident at CERN, and a collection per merged AOD file in ROOT format, using the tag definitions implemented by the physics analysis tools group with one added field for scalability tests.

Per-file collections are useful in that one can navigate directly to selected events or to specific event number ranges. Current ROOT-based event collections can be browsed and analyzed directly in ROOT. Tools also exist to convert POOL event collections into AIDA NTuple format. Some sample ROOT macros for making selections would be needed to understand the access patterns and determine an interface which can be used with SQL- and ROOT-based collections in a transparent and efficient manner. Concatenation (or merging) of AOD files is also desirable to avoid the proliferation of files — back navigation does work from merged AOD files. The database group will optimize the back navigation infrastructure so that one needs not navigate from merged AOD to un-merged AOD before reaching the ESD. Regarding the master tag database, the current tag definition needs to contain sufficient information to be able to distinguish the various Rome data samples. If that is not the case, one could build the tag database as N database-resident collections, one per dataset type, or add an attribute to the tag that specifies the data sample. Utilities for querying the Rome database and creating dedicated AOD with only interesting events are available and have been used to build event collections for a few Rome files as a test of the infrastructure. It was noted that a grid-based initiative for building the tag database is not proposed in the time scale of the Rome workshop: the tag databases at CERN will be remotely accessible and the event collection tools are usable anywhere. Further tests of the tag database deployment, interface, and capabilities are planned for the post-Rome period.

8 Histograms in ATHENA

AIDA provides abstract interfaces for histograms in order to support many technologies: HBOOK, ROOT, JAS [13]. The current AIDA implementation is not ideal. It lacks functionalities, and can be cumbersome to use. This is a combination of a bad design and our lack of control over it — in order to change bits of the AIDA interface, we need access to three different packages, some of which are beyond our control. The PyKernel/PyKHist classes are introduced to register AIDA Histograms to the histogram service in the interactive session and also to provide missing functionalities in the AIDA histogram interface.

We need to seriously investigate whether we should abandon AIDA and use ROOT directly. This has the downside of locking us into a persistency scheme, increasing our dependence on ROOT, and may not be a good solution for the High Level Trigger (HLT) community. Although the HLT community prefers a ROOT API to an AIDA one — which is consistent with the feedback from the physics community — an abstract API rather than a concrete one is more desirable to the HLT community so as to allow for flexibility in the implementation.

In the interactive session, the `reinitialize()` on the events should call the `reinitialize()` of the histogram service if the user so desires, i.e., it is the client's choice to decide which histograms should be reset. AIDA does use ROOT behind the scene but the ATHENA `reinitialize()` knows about one, not the other. It was suggested to investigate the possibility of adding functionalities to the AIDA API so as to get access to the ROOT histogram behind it: this will establish a ROOT-GAUDI histogram symmetry.

9 Fast Simulation Tools

A fast simulation tool, known as the *Atlfast comparator* is being developed. The objective is to tune the fast simulation parameters by making detailed comparison with fully simulated or real data. The comparator therefore allows for the variation and/or the tuning of the individual *Atlfast* smearing parameters. Furthermore, the comparator will enable the selection of different parameterization schemes for individual variations and use them in the smearing. The comparator will not generate new parameterizations. Rather, it will allow the tuning of parameters already provided as input to the fast simulation. This is done by running and re-running the

fast simulation on the AOD MC truth¹ and measuring the goodness of fit between fast and full simulations: fast and full simulation histograms are compared through a χ^2 or Kolmogorov test using Minuit to alter the smearing parameters and feeding them back into Atlfast. However, due to the high number of iterations required for the Minuit minimization, Atlfast is run after the AOD production. This is about a hundred times faster than running Atlfast during the production of the AOD. It is desirable to move the hard-coded Atlfast smearing parameters to job options to vary them during the minimization without recompilation. The minimization algorithm is written in C++ as Python seems too slow for the high number of required iterations and the Minuit minimization was not successfully run in PyROOT.

Some use cases of the comparator include the comparison of full and fast simulation quantities in the AOD; the tuning of Atlfast, etc [14]. From the initial comparison on electrons, it is important to define accurately the quantities to compare and also to make "equivalent" cuts on both fast and full data samples. The core output of the comparator would include vertex distributions, track multiplicities, particle and jet distributions, b-tagging quantities and the development of tracking efficiencies as a function of pseudo rapidity. It was recommended to implement track parameters (TrackParticles) in the fast simulation AOD. The first automated tuning algorithm will be available in the immediate future followed by a prototype (which should include electron smearing and b-tagging parameters), and a production series in the time scale of several months.

It was noted that one should eventually generalize this tool beyond the tuning of just the fast simulation to include for example the tuning of event generators. There should be a book keeping (documentation) on the different parameterizations and on the different parameters.

10 Event View

The idea of an *event view* was proposed at the UCL meeting. At that time it was vaguely defined as "a coherent and an exhaustive list of physics objects that are mutually exclusive". It was acknowledged that the event views are not unique; a user may wish to consider different views of the same event. By coherent, it is meant that the user needs not to carry out additional checks nor call additional tools to guarantee the self-consistency of the view. By exhaustive it was meant that the sum of the

¹The full MC event record is saved in the AOD for this. Atlfast is NOT run on the slimmed MC event record

energies of the objects in the view should come out to be the observed energy in the event and the total sum of the transverse momenta, including the missing transverse momentum, should be roughly zero. Objects in the view should be mutually exclusive: for example, a jet should not be listed also as an electron. Thus, overlap checking (also referred to as redundancy removal, ambiguity resolution, and arbitration) is an integral part of the tools needed to create the event views.

During the workshop we discussed the following in relation to the event view

- the state of overlap in the AOD and the source of that overlap.
- the current approaches to removing this overlap.
- the algorithms and tools needed to produce event views
- a python implementation of a class similar to the event view
- a C++ proposal for an Event View class
- the role of event views as an NTuple-like objects accessible in ROOT and ATHENA.

It was noted that the objects in the default AOD can overlap (be redundant, be ambiguous, etc.). Some of the overlap is removed during the AOD building, e.g., the Electron and Photon collections do not overlap, due to the requirement of a track match (Electron) or not (Photon) on the egamma object of the ESD from which the AOD Electron or Photon is constructed. Similarly, overlap between “soft” and “egamma” Electron candidates as well as high- and low- p_T Muon candidates is removed. However, many other collections of particles do overlap by default as the removal process may be physics analysis dependent. Thus, a set of configurable tools and examples are needed to help the user deal with the situation in the analysis stage. In this respect, the SUSY group has implemented some redundancy removal tools which could be generalized, made available to all users, and utilized in the construction of event views.

An event class used in a Python analysis was presented: it is a collection of named particle lists and functions to operate on the list. One does not write the event class but just the functions and the tools for filtering and selections. The motivation for this event class, styled after the ROOT based analysis in D0, came out of the developer’s own physics analysis exercise in Python as the desire to be able to easily get a handle on intermediate analysis results and cache data in memory in order to

speed up the interactive session. This example of an event view has been tested on sample sizes of the order of 10,000 events. It uses `root_pickle` (instead of `pylcardict`) to persistify histograms and the Python structure, e.g. `h.elec0.pt.Draw()`, in ROOT. The `root_pickle` tool is easier to implement (about 20 lines of code); it is possible to have references to external objects (not just ROOT objects) but references to AOD objects would be harder to implement. It was noted that it is a clever design and implementation of an event class with some similarities with the concept of the event view. Unlike in the event view concept, the algorithms for the redundancy removal, the overlap checking and the ambiguity resolution are not integrated into this event class but they can easily be implemented.

Regarding a C++ implementation, two aspects of the event view were addressed, the event data model (EDM) of the event view and the algorithm and tools to build event view objects. An algorithm to build event views, based on the Expert Systems [15] was discussed, (see Figure 2). The user sets up a Strategy with the type of Actions she wants to take for each Rule which is defined (by the event view Manager). Each Rule has a set of Conditions (e.g., has a matched track) as well as a set of Consequences (e.g., maybe an Electron). Then all the Criteria need to be satisfied (e.g., thrust cut, isolation, ...). The algorithm machinery is hidden from the user and encapsulated in the Engine. The event view Manager should have two modus operandi :

1. Check that some hypothesis can be verified, and return accordingly (the best or a set of) event view,
2. Find a default event view from a set of Conditions.

However, the detailed implementation of this Expert Systems based event view algorithm needs to be worked before the feasibility and usefulness of the approach can be established. The tools used by the algorithm to build the event view, which in the Expert Systems approach would correspond to the Rules with their sets of Conditions, include the redundancy removal, the overlap checking and the ambiguity resolution.

Regarding the EDM component of the event view, a model was presented in which the event view provided two major functionalities:

- The event view as a proxy for particle iterators in the sense that one can ask the event view for the list of all the particles that make up that particular view or for the list of all the Electrons in that view. As a result, `SymLink`

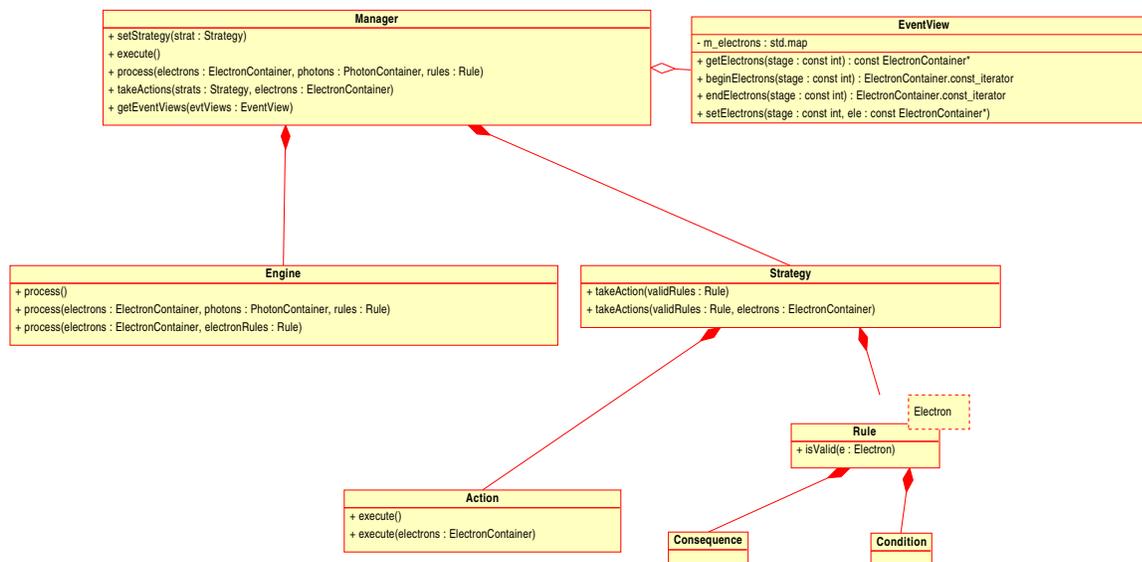


Figure 2: A UML class diagram of an expert system, which could be used as a framework to build event view objects.

would be restricted to navigation, not needed for typical user's analysis. The event view could have master iterators across the different particle containers. It was cautioned that the event view should not duplicate the functionalities of StoreGate nor become a mini-StoreGate.

- The event view could have the ability to store some user data, such as sphericity, Higgs mass, etc, for that particular view of the event. The event view, with the user data, would be persistifiable in such a way that the user data of the event view would be accessible in ROOT as an NTuple-like object and could be read back into the ATHENA framework. At the same time, the event view would maintain the hooks and links (which are not necessarily understood by ROOT) back to the original AOD in order to access the AOD during an interactive session for instance.

A detailed design of an event view class was presented and discussed. The design is shown in Figure 3. A working prototype of this design has since been created and appears quite promising. Discussions are ongoing to provide the desired ROOT and ATHENA compatibility.

While the Event View class is being developed, it was agreed that the algorithm that creates the event view should also write out the view containers of the final state particles (mutually exclusive and non overlapping objects) to StoreGate, i.e., a global

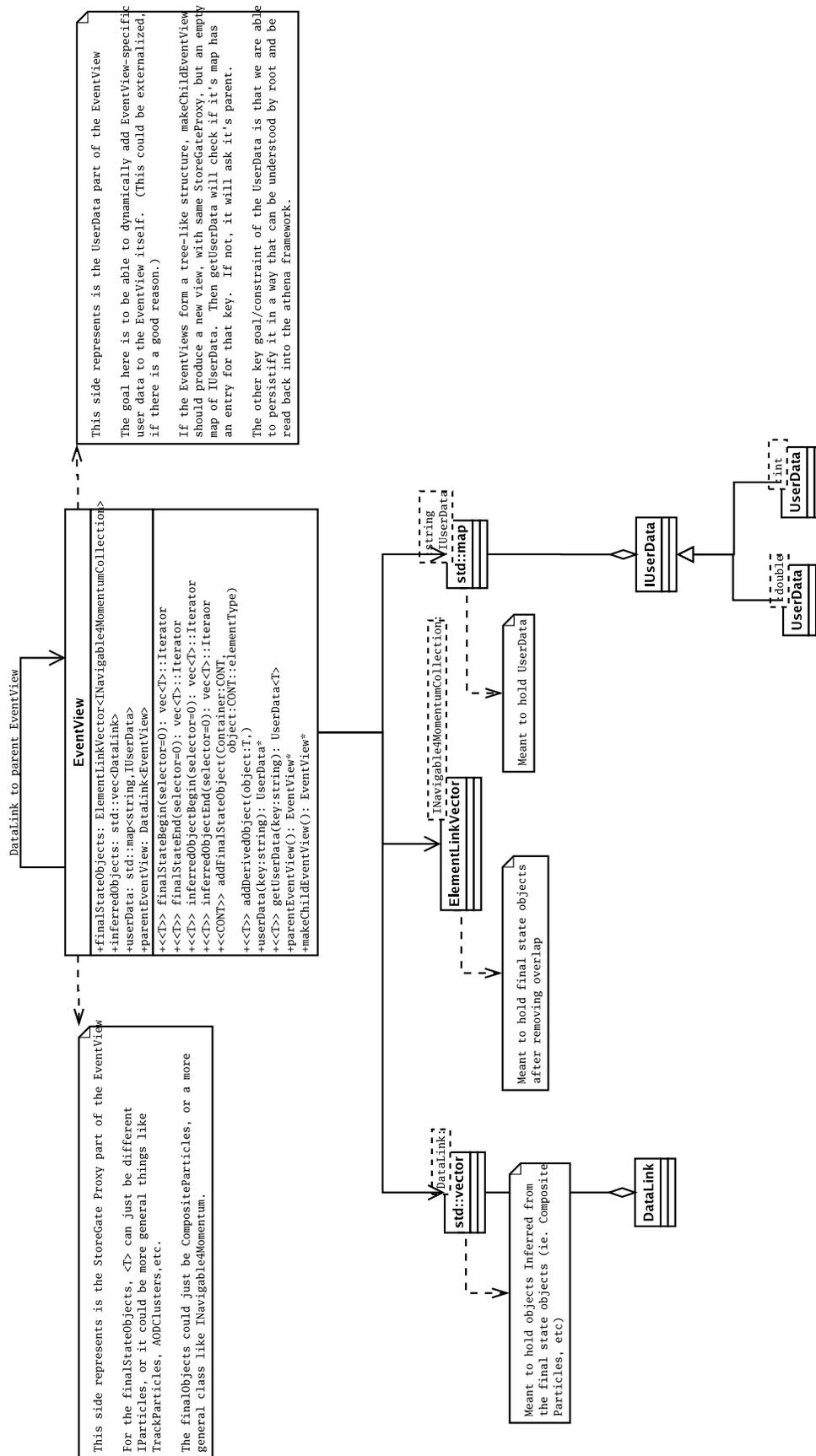


Figure 3: A UML class diagram of one Event View class proposed at Tucson.

data bucket over which the master iterators can operate.

Acknowledgment

Special thanks to P. Loch and J. Rutherford for hosting this workshop at the University of Arizona, Tucson, AZ, USA.

References

- [1] UCL Workshop Summary Report, www.usatlas.bnl.gov/PAT/ucl_workshop_summary.pdf
- [2] AOD/ESD Definition Task Force Report, ATL-SOFT-2004-006, (2004)
- [3] ATHENA, <http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/>
- [4] ROOT, <http://root.cern.ch/>
- [5] <http://www.python.org/doc/>
- [6] POOL, <http://lcgapp.cern.ch/project/persist/>
- [7] <https://uimon.cern.ch/twiki/bin/view/Atlas/ProjectReleases>
- [8] SEAL, <http://seal.cern.ch>
- [9] CINT, <http://root.cern.ch/root/Cint.html>; The SEAL C++ Reflection System Computing in High Energy Physics (CHEP04), CH-Interlaken, Sept. 27-Oct. 1, 2004
- [10] PyROOT, <http://wlav.home.cern.ch/wlav/pyroot/>
- [11] CLHEP, <http://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/>
- [12] Geant4, <http://wwwasd.web.cern.ch/wwwasd/geant4/geant4.html>
- [13] JAS, <http://www-sldnt.slac.stanford.edu/jas/>
- [14] ATL-SOFT-INT-2005-002
- [15] Expert Systems, <http://freelock.com/technical/expert.php>