Overview of ATLAS Database Activities

A. (RD) Schaffer, representing the ATLAS software team
CERN, Geneva, Switzerland[*]

1 Introduction

Database work has begun in a number of areas within ATLAS:

- offline:
  - Detector Description domain, and
  - Event domain
- Test beam
- Detector construction measurements

Each of these areas will be covered in turn. We are in the early stages of designing the various systems that need to be stored, so that we interpret the term "database work" in a fairly broad sense of system design. And we are not strictly limiting the discussion storage-related aspect. The current baseline for ATLAS is to use an object-orient database management system (ODBMS) for its major storage needs: i.e. event data, calibration/alignment, and detector description.

2 Offline: near term goals

One of our major near term goals is to have a first version of an object-oriented simulation and reconstruction framework available by the end of 1999. Since all of the work for the design of the detector has been done using FORTRAN, this means that we must move both the software AND the programmers to C++. One of the key first steps is to provide a C++ framework for people to develop within. We have adopted a framework initially written by Toby Burnett, called Arve. (This will not be discussed further here.) Another key point is to provide access to existing simulated data, which is needed for any realistic test of newly developed code.

2.1 Offline: Detector Description domain

Detector description refers to all the parameters needed to describe the detector, in terms of material, shape, position, etc., for all offline applications. One of the key requirements that we have is to provide a *single* source of detector description information for *all* applications, as can be viewed in Fig. 1. Here there is a single detector description data store which is considered the master source. In the near term future, this will most likely be simple ASCII files, and in the longer term and ODBMS. The exact connection with the CAD system is today uncertain. Although it is desired to be able to use the same information that the engineers use, the technological connection is not yet clear. Within the context of Geant3, this has been possible with a product which runs on a Euclid CAD system. But work remains to be done to explore the possibilities with Geant4 and, for example, the use of its STEP-Express interface.

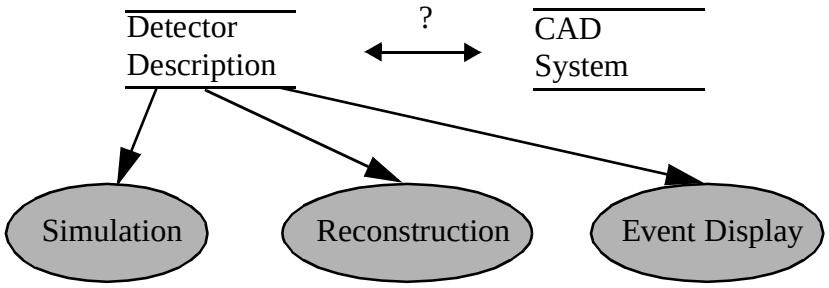

Figure 1: Detector description database is the primary source for all applications.

Our detector description work has begun by concentrating on an existing "database" developed for the muon spectrometer. This database is simply an ASCII file which contains the description of the different types of chambers and as well the positioning of the actual elements which will be built. Some thought had gone into this description to allow it to be concise, for example by allowing a simple parameterization for the chamber positioning in phi which exploits the phi symmetry. An object model has been easily derivable from the description. The basic functionality of the ASCII file database system is to read the file and fill either FORTRAN commons or Zebra banks. This functionality has been preserved.

In developing an object oriented version of this muon database, two aspects are worth mentioning: the overall architecture and the use of hierarchical identifiers. The basic architecture can be viewed in Fig. 2. Here there is a central transient object model for the general

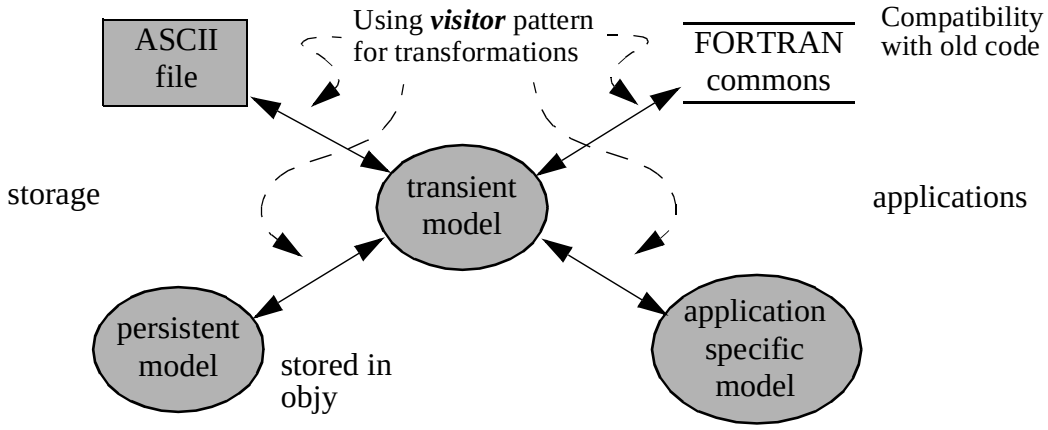

Figure 2: Overall architecture of the detector description domain

description. All other models are filled from this central transient one. For example, there are the application specific models for the simulation and reconstruction. The central model must contain sufficient information for the different requirements of the various applications. Note that the central model decouples the applications from the actual persistent storage mechanism. In this way, one can easily evolve from an ASCII file being the primary store to a full-fledged object database. A generalised mechanism for transformations between the different representations uses the visitor pattern (see ref. [1] for a discussion on visitors).

The use of hierarchical decomposition, i.e. a part contains other parts, etc., is a fairly natural way to describe a detector. For those familiar with Geant3, this technique is used to build a geometry tree which in turn is used for the tracking. Any node in a tree can be identified by its path from the root. We have adopted such a scheme for identifying detector parts using

This general identifier class is used to identify the parts of the detector, e.g. each read-out channel, in a logical manner. For example, the ATLAS Pixel vertex detector is composed of wafers which can be viewed as being logically organized into barrel or endcap, layers, rings and phi sectors, i.e.:

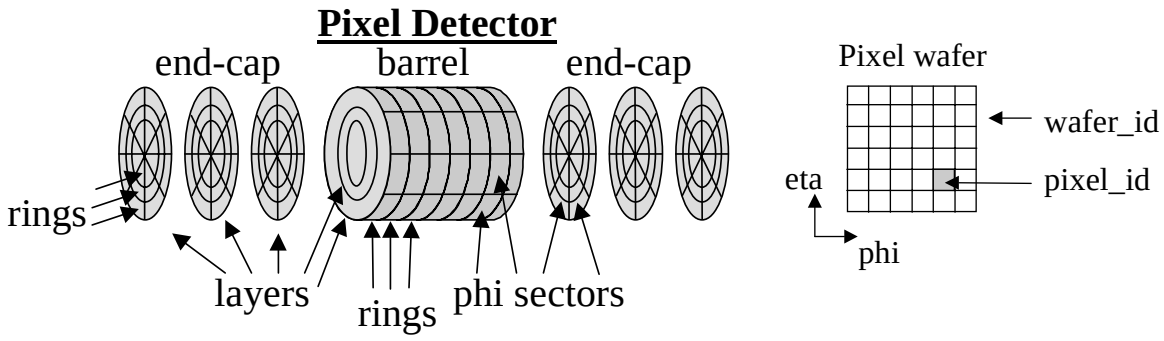

Figure 3: Logical decomposition of the ATLAS pixel detector

Thus, an individual wafer may be identified as: wafer_id = "barrel / layer 1 / ring 3 / phi sector 5", and an individual pixel: pixel_id = wafer_id + "eta 19 / phi 33"[2]. These identifiers can be used as a means mapping detector-related information together, for example from raw data to calibration or detector description quantities. They can also be used to select data, for example by reconstruction algorithms. In order to help with selection, a Range class has been developed which allows a concise expression of a range of identifiers. For example, in Fig. 4 the rectangular set of pixels may be specified by a Range object which has been initialized with the pixel identifiers at the two corners.

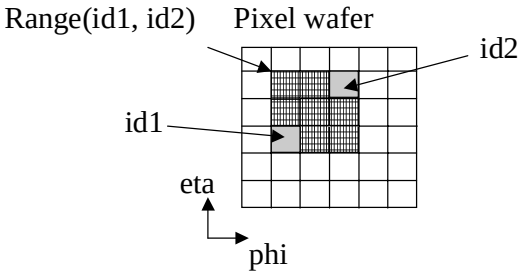

Figure 4: Identifying a set of pixels with a single range object

A working transient-only version of the detector description database is now available for part of the Muon Spectrometer to allow feedback from clients. Our next steps are to add persistency and begin to extend to other detector systems. As a final point, this work within ATLAS is not going on in isolation: an informal collaboration has recently begun at CERN between the LHC experiments. It is not yet clear what common elements will be the result of this effort, but already the exchange ideas has been found useful.

[1] The formatting information defines which bits in a bit string provide the number of each node in the identifier tree. The current implementation stores a pair of numbers for each level in the identifier tree: a number indi-

2.2 Offline: Event domain

Given the above mentioned near term goals, the primary effort in the event domain has been focused on the design and implementation of those classes needed to access the raw data generated by the current Geant3 simulation. For the moment, we have chosen to separate the transient and persistent parts of the software, similar to BaBar's approach. This is particularly appropriate for raw data where there is a large amount of data, and one may want to decouple the question of disk storage from the actual appearance of the objects in memory. Currently, we are able to provide access to the raw data for most of the detector systems, and the storage of raw data in Objectivity is just now becoming available.

The basic design of the raw data structure can be seen in the class diagram of Fig. 5:

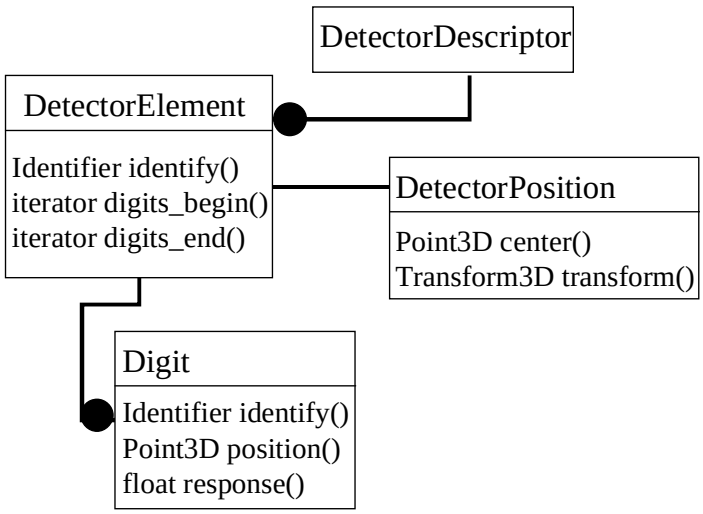

Figure 5: Class diagram of the raw data structure

Here the raw data, or digits, are always organised by a "containing" detector element. In the case of the pixel detector example above, a detector element corresponds to a single wafer[3]. Each detector element is "Identifiable", i.e. provides an identifier, to allow for both identification and data selection. The digit objects are accessed via the detector element objects using STL-like iterators[4]. This design allows digits to provide not only "raw" data, e.g. strip or pixel number, but also more useable "physics" quantities, e.g. local and global position or drift time and drift distance. The transformation from "raw" to "physics" quantities is provided by a detector description object and a detector position object attached to the detector element object, where the former knows, for example, about changing channel numbers to local positions and the latter can be used to transform to global coordinates. Finally, this design allows the flexibility to adjust the amount of physical memory storage consumed by the digit, which can be important in ATLAS.

The detector elements are organised into a tree structure which is connected to an event object. The user interface provides "collector" objects which visit this structure and collect either detector element objects or the digit objects themselves. Clients may select subsets of the detector element or digit objects using the above-mentioned range objects. This would be typically useful when reconstructing within a roads.

[3] The "granularity" of a detector element may of course vary with the detector system. It may correspond to a whole multi-layer chamber or just part of a single layer. The key criteria here is to identify the granularity

The question of the separation of the transient and persistent parts of a event has not yet been fully answered within ATLAS. For the raw data, we have chosen to do so for the following reasons:

1. can have multiple data sources - e.g. Zebra/Objy,
2. memory/disk space considerations can be treated separately, and
3. eventually, the online filter farm (3rd level trigger) will want to use offline code and thus need to use the same event interface to access the raw data coming out of the trigger.

Although partially answered, the question of transient/persistent separation must be treated for the full event data.

Concerning the overall event design, we are only beginning to attack this question. Some of the basic questions that need to be answered are:

1. Who sees persistent objects? Can one hide the persistent/transient difference?
2. How do clients select/identify/navigate to objects in events?

Our basic ideas are similar to BaBar's - storage according to the various levels of detail, e.g. raw data, event summary data and analysis object data, and perhaps access via physics-oriented groups, e.g. particles, tracks, EM clusters, etc. Some of the basic considerations are:

1. avoid schema evolution of the basic event model
2. provide a simple API
3. allow mapping to the event filter where only a transient event model will exist.

We will answer these questions with the help of the work and examples of RD45, BaBar, D0 and others.

Finally, we plan to "exercise" the use of Objectivity by writing ~1 TB of Geant3 raw data and analysis "ntuples" into a system with 100 GB of disk and staging to tape via HPSS by the end of the year. This will allow us to make various benchmarks using ATLAS data.

3 Test beam use of Objectivity

The first ATLAS experience with using Objectivity in test beams has come this year with the Tile Calorimeter test beam. In only a six-week period, a group of about two people with little experience with Objectivity moved their C/Zebra-based event store to an object-oriented system using Objy V5. They made no major redesign of the basic C-structures of their events. They wrote >100 GB of event data in parallel to the standard Zebra system during a summer test beam period. The reconstruction programs are being developed since the data-taking. They did redesign their calibration system - they developed an hierarchical detector model which is derived from a cabling description[5] and is used to access different sets of calibration constants.

Other groups will begin using Objectivity for next year's test beams. In particular, we would like to try out the use of BaBar's Conditions database which is being adapted for general use by RD45.

4 Detector construction measurements

A small project has been started for the construction phase of the Pixel detector system. They would like to maintain in a database the processing history and the set of measurements

performed on each device. They would also like to have a web interface to the information. They are using the Objectivity Java binding. They have completed their design and are working on the implementation. Their first tests will be in October 1998.

5 Summary

ATLAS has made good progress in providing access to the detector description and raw data for object-oriented offline developments. We are just now beginning to see the first use of these new elements. We are still early in developing our expertise with Objectivity with ATLAS, but this is growing rapidly.

6 References

[1] E. Gamma, et al., Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley