# Testbeams and Testbeds: Approaches to Object-oriented Data Access in ATLAS

*David Malon*[1] *for the ATLAS Tile Calorimeter Software Team*

Argonne National Laboratory, Argonne, IL, USA

**Abstract**

In the summer of 1999, ATLAS undertook a pilot project to develop object-oriented software that would simultaneously support tile calorimeter testbeam data analysis and provide a testbed for evaluation of candidate ATLAS software strategies. Strategies related to object persistence and to transient/persistent separation were of particular interest.

Among the distinctive features of the resulting architecture are its detector-centric view, which allows physicists to focus on the detector when appropriate and permits the event model to remain opaque, and its ability to support multiple distinct transient/persistent mapping strategies concurrently. The software has been successfully applied to analysis of 1998 and 1999 ATLAS tile calorimeter testbeam data, with both input and output data stored in an object database.[⋆]

Keywords: object persistence, databases, detector models

## 1 Background

In the spring of 1999, a review of ATLAS software produced a recommendation that ATLAS adopt a software strategy that would keep physics codes, insofar as possible, independent of database supplier, while not altering the ATLAS baseline database strategy, which was (and is) to use the Objectivity/DB commercial object-oriented database product. ATLAS had by that time accrued some experience in the use of Objectivity, including a one-terabyte test in December 1998, but Objectivity had not yet been used in an ATLAS production setting. At the same time, S. Solodkhov (Protvino) had begun an effort to store raw data from ATLAS tile calorimeter testbeam runs in an Objectivity database.

With these developments in mind, a pilot project was undertaken to use tile calorimeter testbeam data for a variety of purposes: as a means to gain production experience with Objectivity, as a testbed for ATLAS software strategies, particularly for approaches to transient/persistent mapping, and as a way to involve physicists in the use (and development) of object-oriented software. The time to accomplish this was somewhat less than two months (tile testbeam data-taking resumed in July 1999), and a specific requirement was that the software be able to support the activities (calibration and analysis) of tile testbeam physicists.

## 2 Software Functionality

The pilot project software provides an architecture for access to data acquired during the 1998 and 1999 ATLAS tile calorimeter testbeam runs. It serves as a tool for data analysis in C++, for

[⋆] The submitted manuscript has been authored by a contractor of the U.S. Government under contract No. W-31-109-Eng-38. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

calibration studies, for optimal filtering, and for building customized n-tuples for later analysis or histogramming via external analysis tools.

Data are stored in an object database. Raw data, calibrated data, and calibration constants (e.g., ADC time samples; energy, charge, transverse energy for a module and any of its constituent elements; timings, charge injection calibration constants) are available through the framework.

For a given run or other event collection, quantities such as a detector element's energy may come from the database if a calibration has already been applied, from dynamic application of a default calibration strategy, or from application of a custom calibration strategy. Physicists may not only read from the database, but may write to it, saving, for example, the results of applying their own custom calibrations as a named event collection.

## 3 Design Principles

*Maintain a detector-centric view.* This is a fundamental design choice, and strongly influences the way in which physicists interact in software with detector and event data. In a *detector-centric* view, the detector has primacy. Events are stimuli to which one wishes to understand the detector's response. In an *event-centric* view, the event has primacy. The role of the detector is to help describe and understand the event. Though these views are duals of one another, and transformations from one view to the other are certainly possible, the difference in emphasis does have implications. For testbeam data analysis, a detector-centric view is natural.

*Keep the event as opaque as possible.* Physicists navigate through the detector, not through the event, to reach and gather data of interest. The detector-centric view allows us, in practice, to keep the event quite opaque–physicists never need to know the shape or content of an event, or how data are organized within the event. This approach is not only consonant with the detector-centric view, but provides the further advantage that software development in the subsystem testbeam community can proceed without fear of significant divergence from ongoing efforts in the core software community to define an event model common to all ATLAS software.

*Describe the detector as a logical hierarchy of identifiable detector elements, using the proposed ATLAS hierarchical identifier scheme.* In the tile calorimeter, for example, the barrel consists of modules (in $\phi$) which are divided into sides, which are further divided into towers (by pseudo-rapidity), in which there are several cells (sampling layers) read out by PMTs connected in turn to ADCs. A unique, variable-length sequence of integers based on ATLAS conventions can be used to identify any element in this hierarchy; this identifier can be stored compactly and managed efficiently by means of an ATLAS Identifier class.

*Detector elements are instantiated only on demand.* Instantiating a calorimeter module, for example, does not require instantiating every side, tower, cell, PMT, and ADC therein. This has performance implications.

*Maintain transient/persistent separation.* Physicists see only a transient model of calorimeter, event, and other data. How these data are organized in the underlying database is never exposed to the client; moreover, references to persistent data objects, whether Objectivity-specific (like ooHandles) or not (like HepODBMS HepRefs) do not appear in physics programs. This strategy is consonant with ATLAS review committee recommendations described above.

## 4 Examples

The following code fragment illustrates the detector-centric view presented by the pilot project software.

```
  TileEventIterator iter;
  for (iter = myRun.begin(); iter != myRun.end(); ++iter) {
//  stimulate the calorimeter with the current event
    myCalorimeter.associate_event(*iter);
    std::cout<<"PMT "<<(pmt1->id())<<" energy is ";
    std::cout<<"pmt1->energy()<<" , timing is "<<pmt->timing()<<endl;
    std::cout<<"Cell "<<(cell3->id())<<" energy is ";
    std::cout<<cell3->energy()<<endl;
//  and so on for any module, tower, cell, pmt, adc, ...
  }
```

Runs support an event collection interface, and provide event-at-a-time access by means of an iterator with the characteristics of an STL (C++ Standard Template Library) forward iterator. Detector elements of interest to the testbeam data analyst–individual PMTs, ADCs, and so on–are typically instantiated once only outside the event loop. Associating the current event with the calorimeter has the same effect as an implicit update of the states of all detector elements–an instance of a PMT object will return the energy corresponding to the current event. The physicist does not need to interact with the event object in any way to navigate to the data corresponding to a particular detector element.

While raw data (e.g., multiple time samples per ADC), calibrated data (e.g., PMT energies), and calibration data (e.g., charge injection calibration constants per ADC) are in fact stored quite differently and in different parts of the underlying database, access to such data follows exactly the same pattern: the physicist simply invokes a given TileADC's samples() or energy() or cis_calib() method in the middle of an event loop, and retrieves the corresponding data.

Comparing calibrations is no more difficult. Default calibration strategy objects are supplied, and users may also provide their own. (A template of a calibration strategy object, and an example of how to write one's own, are provided with the software release.) All that is required is to associate the calibration strategy object with the calorimeter. The following example illustrates this approach inside an elementary event loop:

```
  TileEventIterator iter;
  for (iter = myRun.begin(); iter != myRun.end(); ++iter) {
//  stimulate the calorimeter with the current event
    myCalorimeter.associate_event(*iter);

//  your favorite calibration strategy
    myCalorimeter.associate_calib_strategy(&strategy1);
    std::cout<<"PMT "<<(pmt1->id())<<" energy is ";
    std::cout<<"pmt1->energy()<<" , timing is "<<pmt->timing()<<endl;

//  an alternative calibration strategy
    myCalorimeter.associate_calib_strategy(&strategy2);
    std::cout<<"PMT "<<(pmt1->id())<<" energy is ";
    std::cout<<"pmt1->energy()<<" , timing is "<<pmt->timing()<<endl;
  }
```

## 5 Transient/Persistent Mapping

Because the software is intended to serve as a testbed for core database technologies, it implements a variety of approaches to transient/persistent mapping concurrently. The most common of these are variations on three-layer and $(2+\epsilon)$-layer architectures, with a transient layer (objects as seen by physicists and their programs), a persistent layer (objects as stored by the database), and a mapping layer in between. In their usual incarnations, transient objects hold exactly one data member–a pointer to the adapter object.

A wide variety of strategies are possible even in such a simple scheme. Adapters may be nothing more than structs containing d_Refs or HepRefs to persistent objects, and forwarding of pointer dereferences may be readily inlined. Adapters may handle one-to-one, many-to-one, and many-to-many mappings (all TileADCs may share a single adapter, for example, with their data aggregated in a single persistent object; a single ADC's raw data and calibration constants may come from different places in the database via a single adapter). Adapters may choose to create transient versions of persistent objects to avoid overly frequent references to the database. Adapters may be polymorphic–different concrete adapters derived from the same adapter base class may offer different implementations, or even point to different data stores.

A full discussion of the scope of transient/persistent mapping strategies explored in this testbed is beyond the scope of this short paper.

## 6 Status

The software described above has been adopted for use by the ATLAS tile calorimeter testbeam community. Extensions are underway to provide access to muon wall and beam data, and to coordinate improvements with evolution of the underlying raw data model. If sufficient manpower can be identified, the software may be adapted for use by ATLAS liquid argon testbeam physicists as well. The data access software continues to be used as a testbed for ATLAS core database technologies.

## 7 Acknowledgments

This project was possible only because of the substantial effort that preceded it–work by Sasha Solodkhov (Protvino) to put raw tile calorimeter testbeam data into Objectivity. The ATLAS tile calorimeter community, led by Marzio Nessi (CERN), provided the requirements that shaped the design. Ed May (Argonne) provided invaluable testing, database administration, and software distribution. Tom LeCompte (Argonne) provided and directed efforts to use this software to feed data to standard analysis programs. RD Schaffer and Christian Arnault (Orsay) were the source of the design and implementation of the ATLAS hierarchical identifiers used in this software.. Bob Wagner (Argonne) has begun the work of integrating this software into the overall framework of ATLAS software releases. The University of Chicago tile group, led by Frank Merritt, has undertaken the job of supporting and extending this software.

## References

1 ATLAS Tile Calorimeter: `http://atlasinfo.cern.ch/Atlas/SUB_DETECTORS/TILE/tilecal.html`
2 ATLAS Computing: `http://www.cern.ch/Atlas/GROUPS/SOFTWARE/OO/`
3 Objectivity/DB: `http://www.objectivity.com`
4 HepODBMS (LHC++): `http://wwwinfo.cern.ch/asd/lhc++/index.html`