AN APPLICATION FRAMEWORK AND DATA MODEL PROTOTYPE FOR THE BABAR EXPERIMENT

D.R. QUARRIE

Lawrence Berkeley National Laboratory, MS 50B-3238, 1 Cyclotron Road, Berkeley, California 94720, USA

F.C. PORTER

Physics Department 356-48, California Institute of Technology, Pasadena, California 91125, USA

The BaBar experiment is under construction, and will do physics with e^+e^- colliding beams in the 10 GeV center-of-mass energy region at the PEP-II accelerator at the Stanford Linear Accelerator Center. This experiment is expected to accumulate of order 10^9 events per calendar year, with first data in 1999. The data must be stored efficiently, and must be easily accessible for multiple and frequent physics analyses. The application framework must accommodate a variety of analysis modules and multiple input/ouput streams. The BaBar collaboration has developed a prototype for the application framework and data access, written in C++ using an object-oriented design philosophy.

1 Introduction

The BaBar experiment will do physics with e^+e^- collisions in the 10 GeV center-of-mass energy region at the PEP-II accelerator at the Stanford Linear Accelerator Center. The principal objectives are CP violation and rare processes in decays of B mesons. BaBar is under construction, with first data anticipated in 1999.

Of order 10^9 events per calendar year are expected, with over 10^8 interesting hadronic events. The data must be stored efficiently, and easily accessible for multiple and frequent analyses. The application framework must be flexible enough to accommodate a variety of analysis modules and multiple input/ouput streams.

The BaBar collaboration[1] has developed a prototype for the application framework and data access, written in C++ using an object oriented design philosophy. The application framework accommodates code from a variety of sources in both online and offline environments. It allows access to event data from code written in either C++ or FORTRAN 90. The data access is based on the Farfalla[2] package.

2 BaBar Application Framework Prototype

The requirements on the framework are that it must accept input from multiple sources, including sequential datafiles, the online event server, and Monte Carlo event generators. It must provide output to multiple destinations, including sequential datafiles, the online event server and a null device. Both the input source and output destination should be selectable at run-time. Events satisfying different physics criteria should be selectable and capable of being routed to different destinations. The framework should allow reconfiguration without recompilation or relinking in order to minimize the turn-around time for "what-if" analyses. It should support both text-based and graphics-based user interfaces and must provide support for code written in either C++ or FORTRAN 90.

2.1 Components of the Framework

The framework is based on the concept of *modules*. A module is a fragment of executable code that has a well-defined interface and performs a well-defined service. The interface is imposed by requiring that each module inherit from an abstract parent class. Generally modules are totally independent of each other, operating purely on the basis of their own internal configuration, data taking run specific information and the input event data. A module might generate new information which might be added to the existing event information or might perform a filter function based on the event characteristics or might perform some statistical operation, integrating the results from multiple events.

Each module will provide an interface to the framework that includes a unique name and functions that will be called at the beginning and end of the job, at the beginning and end of each data taking run (*i.e.*, when the run number changes) and a per event function.

Several types of specialized modules are supported within the framework. These include: Input Modules, acting as the source of data; Output Modules acting as the sink of data and Filter Modules which can terminate or re-direct the subsequent processing of an event based on its filter criteria and the characteristics of the event.

Multiple modules can be combined into a *sequence* having a unique name. A sequence may also include other sequences to provide an arbitrary nesting depth.

A *path* is a list of modules and sequences that begins at the input module and terminates at the output module. The processing for a path may be prematurely terminated by the action of a filter module. Multiple paths are supported, corresponding perhaps to different physics processes.

The concept and the organization of the class library are shown in Figure 1.

2.2 Framework User Interface

The Tcl[3] package is used to supply a textual user interface. All Tcl commands are available, and the framework itself adds several commands by which modules may be associated with sequences and paths *etc.* The framework provides a class library by which user written modules may add their own commands to the user interface, thus providing a mechanism by which the internal adjustable parameters of such modules may be modified at run-time.

The prototype framework does not presently support a graphical user interface.

2.3 Support for C++ and FORTRAN 90

The framework provides direct support for modules written in C++ by an inheritance relationship from the default *Module* class. Modules written in FORTRAN 90 require a C++ wrapper but then have complete access to the event information.

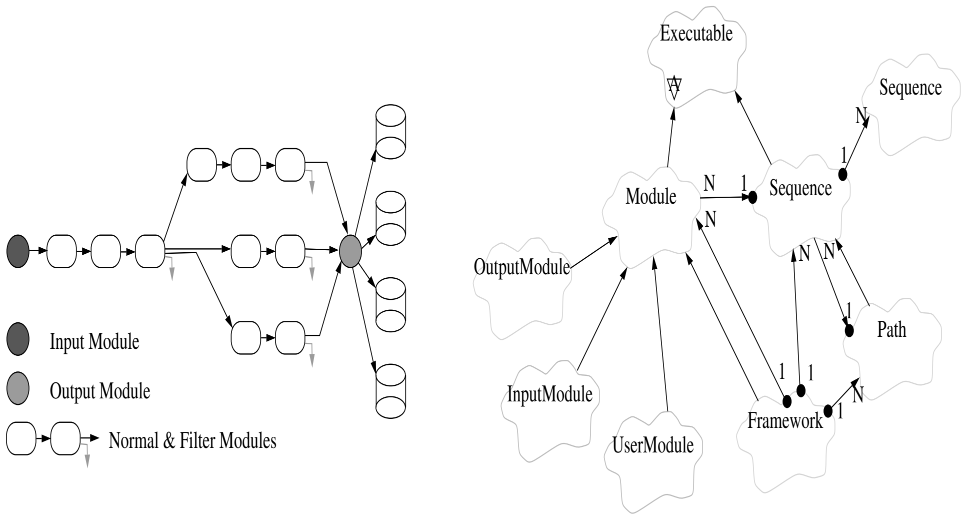

Figure 1: Left: Framework concept; Right: Framework class diagram.

This is provided by a set of interface routines and C++ glue code that is automatically generated from a set of interface definitions written in the Interface Definition Language (IDL) using an IDL to FORTRAN 90 compiler.[4]

3 BaBar Data Model Prototype

A prototype data model, called "Colias", has been created for BaBar, based on the Farfalla[2] package. Both Farfalla and Colias are written in C++. A partial interface to the BaBar GEANT-based Monte Carlo simulation package exists.

3.1 Data Model Description

The data model uses the notion of a "tree", which is a hierarchy of nodes. The nodes are "persistent" objects – that they can be stored and restored. However, it is the nodes themselves which provide the knowledge of what is required for persistence, rather than any abstraction. A node is implemented as a C++ class, the base class in Farfalla is called *F_Node*. To provide for BaBar enhancements to the Farfalla class, an intermediate *DATColiasNode* class is provided, and all Colias nodes inherit from this class. The class hierarchy for a portion of Colias is illustrated in Figure 2.

The "parent-child" association of nodes is maintained by Farfalla. The implementation of a collection (*e.g.*, of *SVTDigi* objects) within a node is currently with a simple array declaration, but we are investigating the use of the STL container classes for this purpose (the CLHEP *HepAList* class has also been used).

There can be more than one kind of "tree" in a dataset, but the most common one will be the event tree, containing the data for one event. There is, however, already a second kind of "tree" which is used to identify the file as a Colias file. Nodes are coded for each sort of data element in BaBar, at a coarse level. Thus

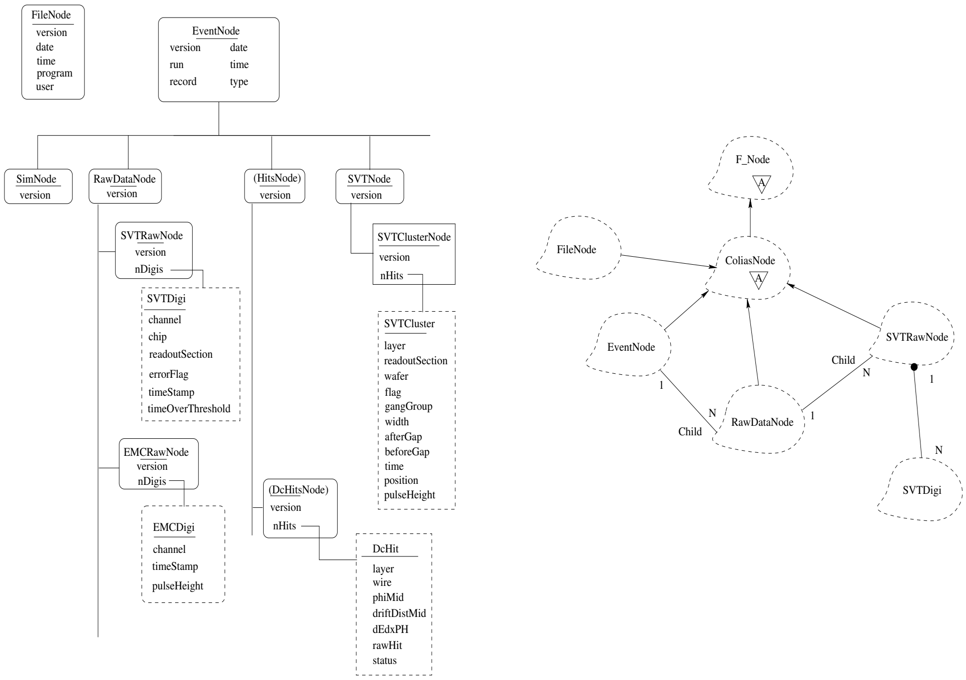

Figure 2: Left: A segment of the Colias prototype data model for BaBar; Right: A segment of the Colias class hierarchy.

there are nodes for "event", "raw data" "calorimeter raw data", *etc.* These nodes are arranged in the form of a "tree" to make up an event record, as shown in Figure 2.

There are several "standard" things that one can ask any node to do:

- Access the data: access to data is via accessor functions, typically in-line for efficiency. The actual data items are encapsulated, permiting changes in the underlying implementation without affecting the user interface. For example, the event number may be obtained by:

```
DATEventNode *event;
....
int eventNumber = event->record();
```

- Modify the data: again, access is via function calls. For example, the event number may be set to 137 by: `event->setRecord(137);`.
- Write to a disk file: typically, this will be done for the whole event tree, with a call to: fstream diskFile; `event->outputSubtree(diskFile);`.
- Reading from a disk file, is accomplished by: `F_inputTree(event,diskFile);`
- Print out summary: for example, a mechanism is provided to print an entire subtree: `event->printSubtree(printFile);`

- Create itself and add to the tree, with a call of the form `F_addChild(event, rawData);`. Here, the first argument is the pointer to the parent node, and the second argument is the (returned) pointer to the node being created.
- Delete itself and free up memory.

3.2 Performance

With large expected datasets, it is important that the data model make efficient use of peripheral storage. Thus, Colias compresses the data in the I/O process. Each node is responsible for its own compression, which is hidden from external view. There is an overhead of 8-12 bytes per node from Farfalla, and an overhead of two bytes per node in Colias to provide flexibility for changes. Because of the overhead, it is not planned that the node structure will be fine-grained, *e.g.*, all calorimeter digitizations are collected under a single node, rather than each in a separate node.

The process of reading and writing the data must be rapid enough to present an acceptable overhead in analysis tasks. Several relevant timing studies have been pursued.[5] There is no firm conclusion at this point, as the interpretation of timing results is complicated by operating system aspects such as the way I/O is buffered in memory. The sensitivity to these complications may be an indication that there are no "order-of-magnitude" excesses in required time.

The data model must be flexible, in particular so that changes can be made, *e.g.*, adding information, without losing the ability to process earlier datasets. Colias provides for this with a "version" member in each node. The version is saved with the node when it is written, and on input is examined to determine how to proceed.

BaBar uses computers from several vendors. At least part of the portability problem is solved by the fact that Farfalla bases its I/O on the XDR "standard".

Acknowledgments

This work was supported in part by the Department of Energy, grant DE-FG03-92-ER40701 and contract DE-AC03-76SF00098.

References

1. BaBar Technical Design Report, SLAC-R-95-457, March 1995.
2. C. Walter and R. Nolty, "The FARFALLA Programming Reference Guide v1.5", July 4, 1994. Available in PostScript on WWW: http://www.slac.-stanford.edu/BFROOT/doc/Computing/farfalla.
3. John K. Ousterhout, Tcl and the Tk Toolkit, Addison-Wesley, 1994.
4. David R. Quarrie, "A Framework for Distributed Mixed Language Scientific Applications", to be presented at this Conference.
5. Timing studies have been made by E. Frank, T. Glanzman, P. Muhl, and S. Saxena, as well as related studies by the authors of Farfalla.