Presented at the Conference on Computing in High-Energy Physics '91, Tsukuba City, Japan, March 11–15, 1991, and to be published in the Proceedings

The EOS TPC Analysis Shell

D.L. Olson

March 1991

Prepared for the U.S. Department of Energy under Contract Number DE-AC03-76SF00098

LBL - 30425

The EOS TPC Analysis Shell

D.L. Olson

Nuclear Science Division
Lawrence Berkeley Laboratory
University of California
Berkeley, Ca 94720, USA

This work was supported by the Director, Office of Energy Research, Division of Nuclear Physics of the Office of High Energy and Nuclear Physics of the U.S. Department of Energy under Contract DE-AC03-76SF00098

The EOS TPC Analysis Shell

D. L. Olson
Nuclear Science Division
Lawrence Berkeley Laboratory
1 Cyclotron Road
Berkeley, CA 94720
USA

Submitted to
Computing in High-Energy Physics '91,
Tsukuba City, Japan
March 11-15, 1991

Abstract

Key features of the general purpose event-based-data analysis shell (TAS) for the EOS TPC at LBL are described including the code development / code management procedures used. The architecture is designed with a view towards a distributed and multi-processing environment. TAS is interfaced seamlessly with the CERN PAW program and provides a consistent environment for both on-line and off-line analysis. The data model used is relational tables and the data structure definitions are maintained in a commercial database (INFORMIX). The interface for analysis modules is specified and enhances group participation in the development process. The use of a commercial database as a data dictionary for both the table definitions and parameters used in the TAS kernel is extremely useful and productive.

1. Introduction

In the course of doing physics research with complex experiments it is common to develop data analysis programs largely from scratch for each experiment. While it is true that a large part of the code necessary for data reduction and physics analysis is unique for each experiment, a fair amount of the interface code is reinvented. There are serious consequences of reinventing interface code; a) the effort required to develop it for each case and b) the effort required for people to learn how to use a new interface. The second reason provided the main motivation for developing the analysis shell used for the EOS TPC[1] at Lawrence Berkeley Laboratory. Both the interactive user interface and the program development interface are designed to enhance program development and data analysis in a group environment.

An overview of TAS is given in the following section including a description of those aspects of the architecture which are designed for distributed event processing and multi-processing. This is followed by a description of the data structures used (called tables) and the methods for describing and managing them. Section 4 describes the code management procedures used. Section 5 describes the use of a commercial relational database (INFORMIX) as a tool for managing parameter definitions and initial value assignments for data structures used in the development of the analysis shell.

2. Overview of TAS

TAS is a general purpose analysis program for event-based data processing. It is designed to handle many of the difficulties arising when doing analysis in a group environment. The major features contributing to this are:

- a specification for connecting analysis routines (analysis modules),
- a separation of the control and I/O functions from the analysis routines,
- a single point of input for specifying the data structures (tables),
- automatic initialization of the data structures,
- and a familiar user interface (KUIP/PAW[2]).

The specification for analysis modules along with the control and I/O functions being provided externally for these routines enables different analysis modules to be developed independently and then easily merged into a single analysis program. A single point of input for specifying the table data structures minimizes the chance for inconsistent definitions. The automatic initialization with default values, specified at the time the tables are defined, minimizes the need for file input simply to make the program run. By choosing a user interface which has many on-line help features, and that has become widely used, minimizes the time needed by most people to learn how to use this analysis shell.

A schematic diagram of TAS is given in Figure 1. There are three separable units in TAS, the user interface (UI), the processing manager (PM) and the event processor (EP). The user interface unit consists of KUIP command routines that format data structures for a message-response command distribution mechanism that communicates with the processing manager unit. The processing manager handles data I/O functions, event processing control, histogram booking and filling, and communication with the event processor. The event processor unit contains the experiment specific analysis modules that access data in the event-buffer tables.

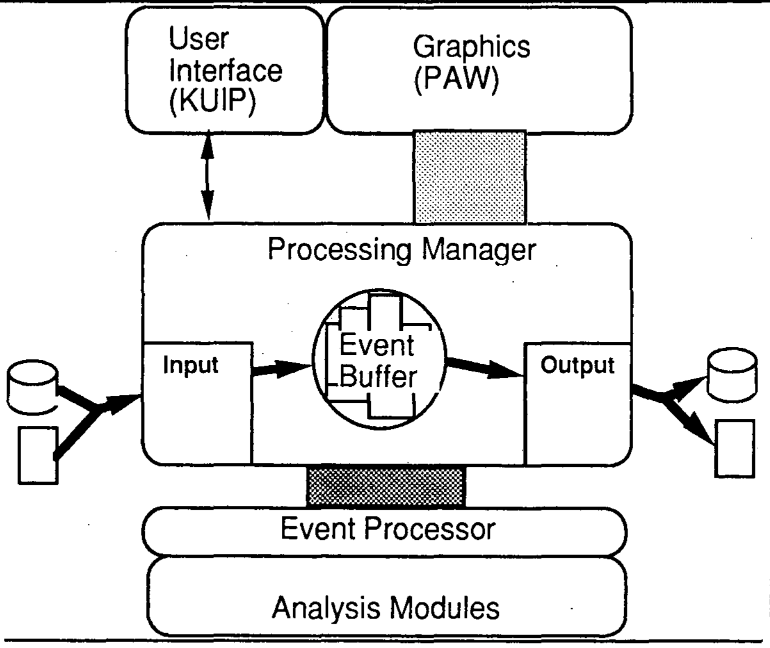

Figure 1. An overview of a TAS application program. Experiment specific routines are in the analysis modules section.

The design is such that these three units run in a single process (in the first release) or can be split into a separate process for each unit (in a later release). The single process mode is very useful for debugging purposes and is envisioned as always being available. The communication between the UI and PM processes is seen as a one-to-one connection where the command link has been implemented as a message-response mechanism (via mailboxes on VAX/VMS). The inter-process connection for graphics is shared memory so that one always has the UI and PM processes executing on the same machine. The inter-process PM-EP link is seen as a one-to-many connection where a single PM process handles distributing events to many EP processes. This is to be implemented as a message-response command and data channel to enable EP processes to be distributed on a network. With this type of implementation the EP processes can equally well be distributed among multiple processors within the same machine.

The UI-PM asynchronous link is developed because of its value in an on-line environment where one wants to be accumulating histograms and viewing them at the same time. The PM-EP link for distributed event processing is seen as appropriate for off-line data processing so one can take advantage of many nodes on the network in addition to multiple cpu's in one machine.

A detailed view of the processing manager unit is given in Figure 2. Commands are received from the UI and distributed locally or sent to the EP.

A user entry is called at a number of different stages of program execution, as indicated in Fig 2. A flag identifying the stage of execution and the event buffer are provided to this routine each time it is called.

The data I/O is performed by TAS routines. The source code that depends upon the particular definitions of the table data-structures is generated automatically by a database procedure.

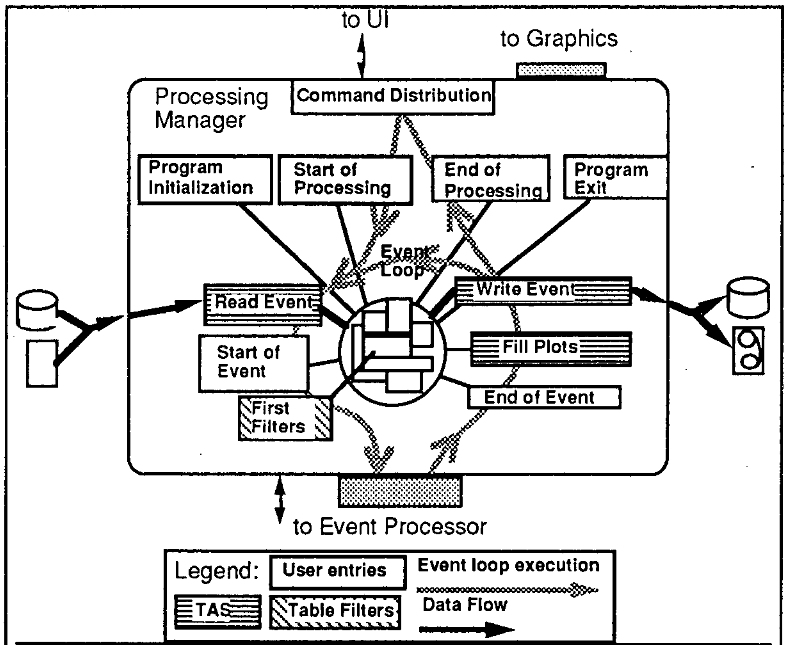

Figure 2. A functional view of the processing manager unit showing user entries and the I/O parts of the event loop.

The HBOOK[3] histograming package is used for plots. Plots can be defined interactively and TAS will automatically fill the plots from the tables in the event buffer at the end of processing each event. PAW is used to view the plots and for analyzing the spectra recorded by TAS.

Table filter routines are executed by the COMIS[4] interpreter and can consist of either compiled or interpreted routines. They access a single table in the event buffer and can modify or delete parts of the table, or affect program execution based upon a return status. The time at which a filter is called during the event loop is specified as part of the filter definition. Filters which are called before any analysis modules are called "First" filters and these are executed in the PM. These are called from the PM to enable cuts on the data that is passed to the EP unit. Additional filters are executed in the EP.

Figure 3 shows the EP, consisting primarily of analysis modules. Analysis modules are declared at initialization time along with the access mode they have for a set of tables. The access mode is either read or write access, depending upon whether or not the analysis module will only read from the table or write into the table.

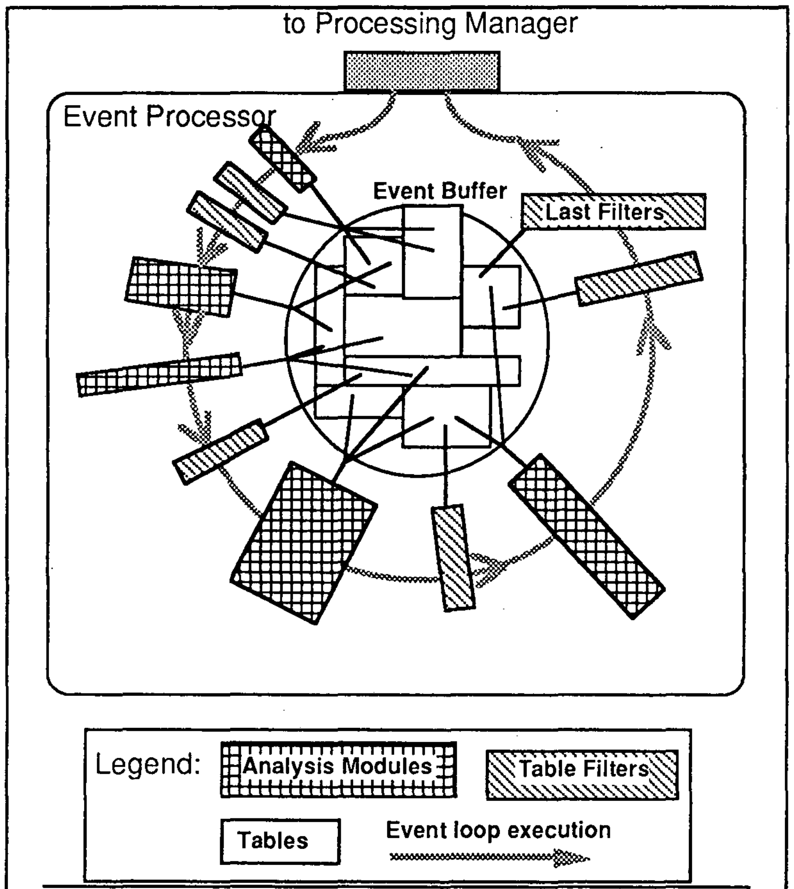

Figure 3. Diagram of event processor showing analysis modules and table filters.

When the EP gets an event it compares the list of input tables to the list of tables requested for output (for either plot filling or saving into an output file) and uses the declared analysis module - table dependencies to determine which analysis modules to call in order to process the current event. This automatic mode of calling analysis modules can be over-ridden by manual switch settings to either force or inhibit calling of a particular analysis module.

The table filters can be defined and moved around interactively. They are specified as being called after a particular analysis module. Filters which are specified to be called after the last analysis module are said to be "Last" filters. The "Last" filters are called in the EP, unlike the "First" filters which are called in the PM. This is to permit cuts on the amount of data that is returned to the PM.

3. TAS tables, description and management

The basic data structure used by TAS is a table. An event is described in terms of a set of tables. These are similar to the tables of a relational database or to those of ADAMO[5]. Each table consists of a set of rows and each row consists of a set of variables. TAS variables can be either integer or floating point, scalars or vectors.

TAS tables are defined to be either of static or dynamic types. Dynamic tables are cleared and refilled for each event while static tables are never cleared by TAS itself. Essentially, dynamic tables are for event data while static tables are for calibration constants and switches.

The method of defining a table, the variables, initial values, type, etc., is to use an INFORMIX screen form. One can enter or modify the table definitions easily and the screen form procedure provides a high degree of verification that will catch a number of mistakes as they occur. A set of database reports are used to automatically generate source code for the data structure definitions, initialization, and table-dependent I/O routines.

4. Program development and code management

Figure 4 shows an outline of the program-development and code-management methods used for the EOS TPC analysis. Definitions of tables are entered in the TAS Editor (an INFORMIX screen form). A number of Fortran source files are generated as reports. These source files are then entered into the CMS library which serves as the primary source code archive.

The set of source files that are compiled into an object library are identified by a short list of groups in CMS libraries. There are two procedures which are driven by this list of CMS groups. The first generates MMS[6] descriptions files that contain all of the include file dependencies and the second executes these MMS procedures which update an object library to be current with the newest source code. The first procedure needs to be executed only when new source files are added to the library. The second procedure is executed whenever there is a modification to some of the sources.

The final step in program development is linking the executable image. A standard procedure for this is maintained in the EOS group library and it serves as a reference for individuals who link private versions of the analysis program as part of developing separate analysis modules.

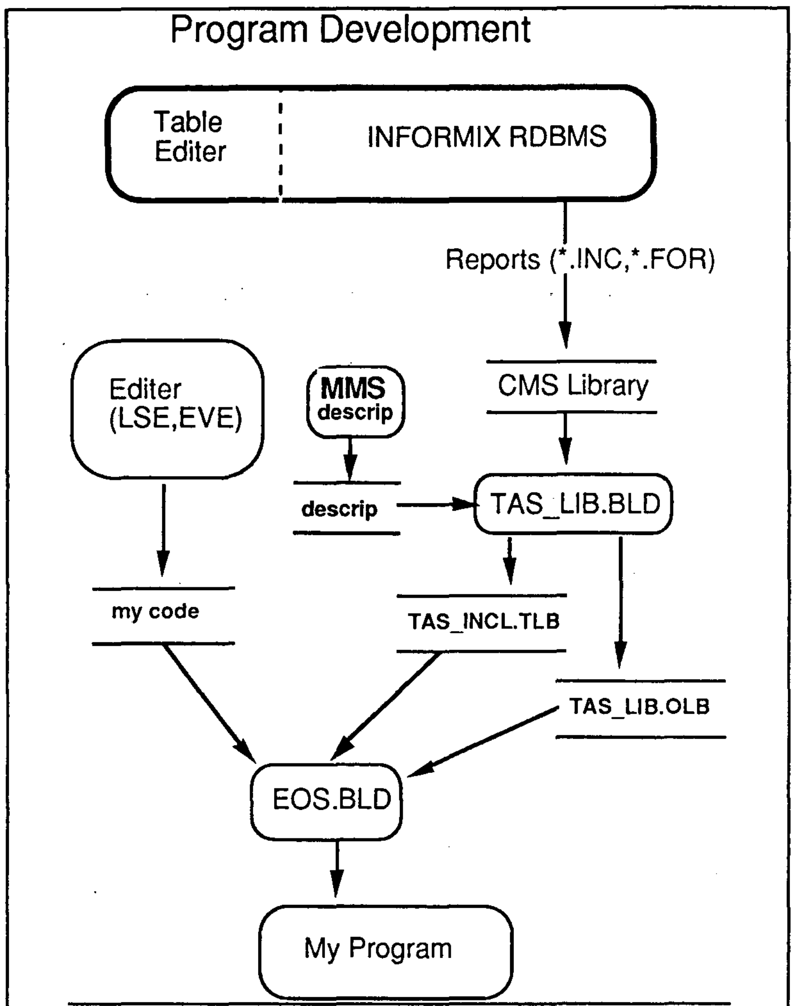

Figure 4. Outline of the program development procedures.

5. Use of INFORMIX DMBS as a code development tool

As mentioned above, the commercial database product from INFORMIX is used during the development of an analysis module as an interactive table-definition editor and a code generator for table-dependent source files. It is also used in the program development of the TAS shell itself. In the latter case, it is used for success/error status parameter definitions and TAS internal command definitions.

For the case of status parameters, a screen form is used to specify parameter names and associated messages. The actual parameter values are generated automatically by the database. Reports are used to generate include files of these parameters and also error-number message-translation routines.

For the TAS internal commands, again a screen form is used to specify names, messages and initial values. Reports are used to generate include files and command-structure initialization routines.

As program development tools, I have found that using INFORMIX forms and reports to be very beneficial. The effort required to set them up is very small compared to the effort required to accomplish the same programming tasks without automated tools. In addition, the cost of such a commercial product is not significant compared to the manpower cost to develop similar automated tools in house.

6. Summary

The analysis shell for the EOS TPC has been designed to facilitate analysis-program development in a group environment. It accomplishes this by using a master database for data structure definitions, defining a standard interface for analysis modules, and providing I/O and control functions outside of the specific analysis routines.

The entire program can run as a single process to facilitate debugging. The asynchronous link between the user interface and processing manager units is appropriate in an on-line environment for continuous histograming. The architecture of the shell is such that it can be extended to support distributed event processing with single input and output data channels. This is most useful for off-line data processing to take advantage of many nodes or cpu's.

The use of a commercial relational database as a program development tool, both for developing the analysis shell itself and as part of physics analysis program development has turned out to be fairly easy to implement and and very useful. It saves time for program development and provides mechanisms to minimize many types of mistakes.

Acknowledgements

This work was supported by the Director, Office of Energy Research, Office of High Energy and Nuclear Physics, Nuclear Physics Division of the U.S. Department of Energy under Contract DE-AC03-76SF00098.

---

[1] The EOS TPC is developed for doing particle identification with momentum analysis and dE/dx in heavy ion reactions. It operates by reading out an array of 15,000 pads and is designed to handle multiplicities of up to 200 charged particles per Au+Au event.
[2] KUIP - Kit for an User Interface Package, CERN-DD I202, 1988,
PAW - Physics Analysis Workstation, CERN-DD, Q121, 1989.
[3] HBOOK Version 4, CERN Program Library, Y250, 1987.
[4] A Fortran interpreter used in PAW. COMIS - Compilation and Interpretation System, User's Guide, CERN, L210, 1988. See also the PAW manual.
[5] Aleph Data Model, Aleph Collaboration, CERN.
[6] DEC Module Management System, similar to UNIX make.