# Object Oriented Run Control for the CEBAF Data Acquisition System[1]

David R. Quarrie, Graham Heyes, Edward Jastrzembski, William A. Watson III
Data Acquisition Group
CEBAF MS 12H,
12000 Jefferson Avenue,
Newport News VA 23606, USA

*Abstract*

After an extensive evaluation, the Eiffel object oriented language has been selected for the design and implementation of the run control portion of the CEBAF Data Acquisition System. The OSF/Motif graphical user interface toolkit and DataViews process control system have been incorporated into this framework. We discuss the evaluation process, the status of the implementation and the lessons learned, particularly in the use of object oriented techniques.

## INTRODUCTION

CEBAF, the Continuous Electron Beam Accelerator Facility, is an electron accelerator with an energy of 4 GeV/c presently under construction. It is designed to allow the detailed investigation of the quark structure of the nucleus and will support several experiments operating simultaneously at three end stations. Beam is scheduled to be available in 1994.

The CEBAF data acquisition system is being developed for use in all three experimental halls for systems with a large range of event sizes and data rates. Thus it should cope with up to 10 kHz event rate with up to 10 kBytes per event, whilst also efficiently supporting other experiments having event sizes of less than 1 kByte.These requirements imply that the system must be flexible, robust and extensible for future enhancements. This in turn implies that the user interface and control mechanisms must be flexible and user-friendly.

Major features of the data acquisition system are:

- FASTBUS, VME, VXI and CAMAC readout modules.
- Intelligent readout controllers (ROC).
- Fiber optic links to a high speed data switch offering total throughput of approx. 160 MBytes/sec.
- A VME-based processor farm offering approx. 1000 MIPS of processing power.
- SCSI output devices.
- Unix workstations for control and monitoring.

The conceptual data acquisition system is shown in Fig.1. Given the large range of data rate requirements, some systems might not be equipped with a data switch and will have less processing power in the VME processor farm.

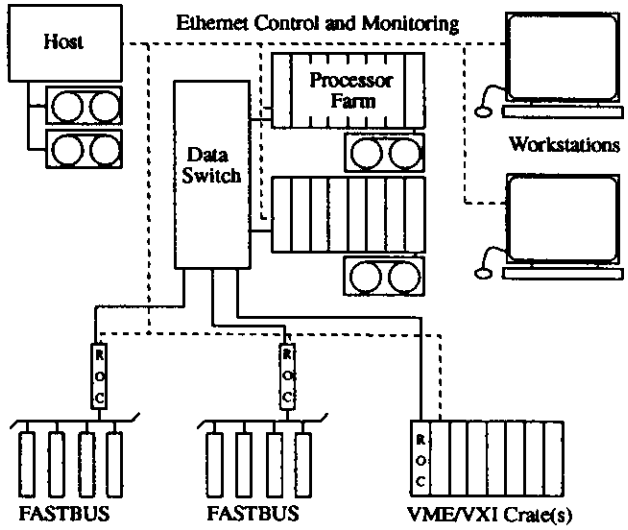


Fig. 1. Conceptual Data Acquisition System

A decision has been made to use object oriented techniques in developing the control software for these systems. After an extensive literature search the Eiffel [1] language and development environment have been chosen as the vehicle for this development. Furthermore, an extensive evaluation of Eiffel was performed prior to embarking on the main project described here.

## EIFFEL

Eiffel is a pure object oriented language, unlike hybrids such as C++ or Objective-C which are based on conventional procedural languages. It is targeted at large software engineering projects requiring a high degree of robustness and low maintenance costs. Major features are:

- A small language having a similar number of reserved words to Pascal.
- Both single and multiple inheritance.
- Static typing combined with dynamic binding.
- Deferred classes to specify behaviour without implementing it.

[1] Work supported by the Department of Energy, contract DE-AC05-84ER40150

### *Draw Program*

This also acted as an exercise in learning X Windows and the OSF/Motif toolkit. The goal was to learn how to create user interfaces in Eiffel, allowing complex manipulations of graphical objects, including dragging them, rescaling them and grouping them into more complex figures. The intention was that such a program would later form the basis of a graphical configuration editor for the data acquisition hardware.

Several important deficiencies in the supplied Eiffel graphical library classes were discovered, many of which could be solved using inheritance, but others, including the lack of figure dynamics such as dragging, could only be solved by direct modification of the supplied source code. Similarly, generation of PostScript code to enable printout of the created diagrams was also lacking.

However, the technique was shown to be suitable for the rapid development of quite complex programs. An example of the capabilities of the resulting graphical editor is shown in Fig.3.

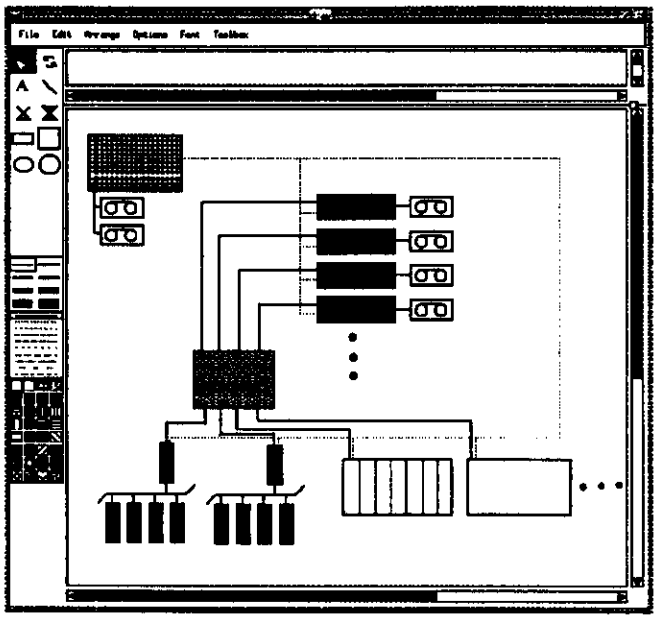

Fig. 3. Example from Draw Program

### *Cross Platform Development*

The VxWorks[1] real time kernel has been chosen for use by both the readout controllers and possibly the processor farm, although in the latter instance Unix based systems are also under consideration. This final evaluation of Eiffel attempted the cross compilation of an Eiffel application (the histogram benchmark described previously) from a Unix host to a VME single board computer running VxWorks.

Eiffel supports the direct generation of C language packages from the original Eiffel source code, but it was felt that use of this technique and subsequent modification of the C source code would not be viable because of the consequent lack of maintainability.

By creating a duplicate copy of the Eiffel class library, replicating a few C header files, creating a small interface library and redefining the Unix environment variable corresponding to the C compiler to reference the cross-compiler instead, this was shown to be possible. This allowed the full power of the Eiffel development environment to be maintained. No change in the Eiffel code was necessary in order to port this program between the two environments.

### *Evaluation Conclusions*

Several quite serious deficiencies were exposed by these evaluation examples. The most severe were the sometimes long turnaround time for an edit/compile/link/run cycle and the lack of a true symbolic debugger, although the ability to inherit from the *VIEWER* class alleviated the latter to a certain extent. However, many of the programming bugs that would have only been found at run time with most conventional languages were caught either by the compiler or through the use of assertions. Assertions proved to be quite expensive in terms of performance, but the facilities provided for enabling or disabling them were easy to use.

Furthermore, it was felt that the graphics classes were substandard, being based on pure X rather than on the higher capabilities of X toolkits such as Motif or Open Look. The standard figure classes also lacked many of the desired features.

However, the major drawbacks are being addressed by the next release of the compiler and it proved to be quite easy to implement a Motif class library. Furthermore, it was felt that the resulting programs were significantly more robust and maintainable than similar program written in conventional languages such as C or FORTRAN. Additionally, these programs were easier to modify and enhance to cope with changing requirements than similar conventional programs. Finally, in most instances where a direct comparison was made, the number of lines of code required to implement a program using Eiffel was significantly less than the equivalent using C or FORTRAN. In combination these indicated that a useful gain in productivity was in fact possible.

It was therefore decided to use Eiffel for the design and implementation of the run control portion of the data acquisition system, conventional programming techniques being used within other areas such as slow controls and within the readout controllers and processor farm.

## OTHER DECISIONS

In parallel with the evaluation of Eiffel, several other software products were evaluated for their suitability within the framework of the overall data acquisition system. The final outcome of these evaluations was the following:

- The use of Unix workstations as both development platforms and control and monitoring platforms.

[1] VxWorks is a registered trademark of Wind River Systems, Inc.

ly the module and it includes the layout of modules within crates and the relationships between slave modules and master modules, together with the hierarchical data paths necessary to access data within each module. One interesting aspect of the object oriented approach is that each module object is itself responsible for performing actions on the corresponding piece of hardware, in contrast to the conventional situation where some central controlling procedure performs the actions.

### *Logical View of Experiment*

This forms a hierarchy of the data acquisition system, its subsystems and components within these subsystems. This simple hierarchy was considered adequate to describe the system, although more complex configurations would be feasible. The following subsystems were identified:

- Readout Controllers. These devices perform the readout of the front end modules. All access to the front end modules, including downloading of calibration constants etc. is via these controllers.
- Trigger System. The details of this will vary from one experiment to another. However, they will all exhibit a programmable interface.
- Trigger Supervisor. This device, the prototype of which is presently being fabricated, is responsible for prescaling different trigger signals and managing the protocol between the hierarchical trigger systems, front end modules and readout controllers. It also manages the multiple event buffers within the front end modules.
- Data Switch. This device is undergoing conceptual design although a final decision on its format will not be made until late in 1992. It is responsible for gathering the event fragments from the various readout controllers and transmitting them as complete events to the several nodes in the processor farm.
- Processor Farm. This will consist of VME single board computers operating either under the VxWorks real time kernel or a Unix system.
- Output Device. Our assumption is that 8mm tape or similar devices will be used.
- Slow Controls. This interfaces the experiment with the CEBAF accelerator and detector status information.
- Data Acquisition Run. This conceptual subsystem describes the behaviour of the data taking run.
- User Subsystem. This includes the user-supplied monitoring and analysis code.

In many instances a component corresponds to a physical module in the physical view above, but in other cases subsystems and components are more abstract, the *DAQ_RUN* object being an example of this. This encapsulates all the attributes of a conventional data acquisition run; the start and stop times, number of events and status etc.

The subsystems within the system are represented as items in a linked list and the components within each subsystem are similarly represented. All components have a unique name and a hash table within the system object allows any component to be referenced efficiently. Thus sending, for example, the "download" message to the *DAQ_SYSTEM* object results in it sending the equivalent message to its subsystems, each of these sending the same message to their components. Use of inheritance makes this a very simple but powerful concept.

### *The User Interface*

The concept of a *MANAGER* is used, where each manager has a target object upon which it acts as a result of operator interaction. Each subsystem has its own manager which is responsible for modifying the configuration and displaying the current status.

Each manager consists of a set of X Windows, DataViews views and Motif menus, panels and buttons. Here again, inheritance was used to reduce to a minimum the amount of code that was written.

### *Summary*

The correlation between the hardware and logical views of the experimental apparatus is implemented as an application of the Multiple Inheritance capabilities of Eiffel. Thus an object can inherit from a parent that encapsulates its hardware behaviour and also inherit from another parent that describes its behaviour within the context of a data taking run. Thus it inherits the concepts of being booted, configured and downloaded etc.

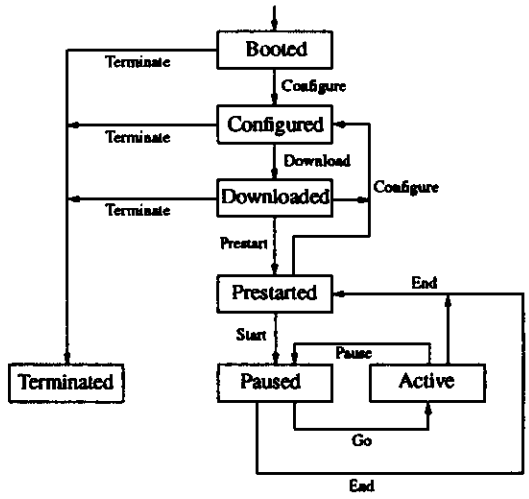


Fig. 4. Simplified State Transition Diagram

A single state transition diagram was found to be adequate to describe the internal states of all logical components; the system, its subsystems and their components. A simplified version of this state transition diagram is shown in Fig. 4. A similar diagram is used to describe the actual hardware.