PHENIX On-Line Distributed Computing System Architecture

Edmond Desmond [a], John Haggerty [a], Hyon Joo Kehayias [a], Thomas Kozlowski [b], Martin L. Purschke [a], Chris Witzig [a]

[a] *Physics Department, Brookhaven National Laboratory, Upton, NY, 11973*

[b] *Physics Division, Los Alamos National Laboratory, Los Alamos, NM 87545*

PHENIX is one of the two large experiments at the Relativistic Heavy Ion Collider (RHIC) currently under construction at Brookhaven National Laboratory. The detector consists of 11 subdetectors, that are further subdivided into 29 units ("granules") that can be operated independently, which includes simultaneous dat a taking with independent data streams and independent triggers.

The detector has 250,000 channels and is read out by front end modules, where the data is buffered in a pipeline while awaiting the level1 trigger decision. Zero suppression and calibration is done after the level1 accept in custom built data collection modules (DCMs) with DSPs before the data is sent to an event buil der (design throughput of 2 Gb/sec) and higher level triggers.

The On-line Computing Systems Group (ONCS) has two responsibilities. Firstly it is responsible for receiving the data from the event builder, routing it through a network of workstations to consumer processes and archiving it at a data rate of 20 MB/sec.

Secondly it is also responsible for the overall configuration, control and operation of the detector and data acquisition chain, which comprises the software integration for several thousand custom built hardware modules. The software must furthermore support the independent operation of the above mentioned granules, which includes the coordination of processes that run in 60-100 VME processors and workstations.

ONCS has adapted the Shlaer-Mellor Object Oriented Methodology for the design of the top layer software. CORBA is used as communication layer between the distributed objects, which are implemented as asynchronous finite state machines. We will give an overview of the PHENIX online system with the main focus on the system architecture, software components and integration tasks of the On-line Computing group ONCS and report on the status of the current prototypes.

Key words: C++, distributed control systems, CORBA,CASE

1 PHENIC Detector Overview

The PHENIX On Line Computing System (ONCS) Software is responsible for the configuration, initialization, operation and diagnostics of the PHENIX detector. The PHENIX detector is one of the high energy physics detectors which will perform experiments at the RHIC Relativistic High Energy Collider facility which is currently being built at Brookhaven National Laboratory. The PHENIX detector is scheduled to begin commissioning operation in the spring of 1999. This paper describes the major components of the ONCS software.
The PHENIX detector consists of 12 separate experiments which will simultaneously exist in one infrastructure. These 12 experiments are listed in Table 1. The 12 PHENIX detectors, while occupying the same facility hall, and physical infrastructure, are physically and logically separate sub detector units. The physical and logical separation of the PHENIX detector determine the operational requirements and constraints on the On Line Computing System. The PHENIX physical plant is split between two large detector arms. Each arm consists of a number of 22 1/2 degree sectors which are themselves consist of separate physical units which are called granules. The detector granules defines the smallest physical unit of the PHENIX detector which can be operated independently of the other PHENIX detector units. There are, at present, 29 defined detector granules with a maximum of 64 possible granules.

Table 1.

Beam-Beam Counter	Electromagnetic Calorimetry (Emcal)
Lead Glass (PbGl)	Lead Scintillator (PbSc)
Time of Flight (TOF)	Ring Imaging Cherenkov Detector (RICH)
Multiplicity and Vertex Dectector MVD	Drift Chamber
Pad Chambers	Time Expansion Chamber
Muon Identifier	Muon Tracking

2 Detector Operational Environment

While the PHENIX detector may be split into these separate physical granules, operationally the granules may be grouped together and operated as a single unit. These operational units of the detector are called partitions of which there can be up to 32 simultaneously in operation. The operation and collection of physics data from one of the partitions constitutes what is commonly thought of as a physics run.
In addition to providing for the operational partitioning of PHENIX detector,

the On Line computing system is responsible for the initialization, configuration and operation of the detector sub systems which are responsible for the construction, routing and ultimate archiving of the physics event data that is collected from the physical detector. These systems are the Data Acquisition system, the Timing system, Level 1 Triggering, sub event buffer, event builder and ancillary systems.
The hardware components of the PHENIX DAQ, Timing, and Level 1 sub systems are resident in VME based modules. These modules are physically distributed over some 60 to 100 VME crates. Each VME crate will contain a separate VME control controller which will be executing the VxWorks operating system from Wind River Systems. The processors will be 680X0 or Power PC based. In addition, the Sub Event Buffers which perform the task of packing and routing the physics event data will be Pentium Pro based machines which will be running the NT operating system from Microsoft. The On Line run control software will be executed from workstations which are physically located in the PHENIX main counting house. However, system monitoring and user application code is expected to run from physically remote locations. This distributed control environment will be supported by the On Line system on a number of diverse platforms including Solaris Sparcstations, and IRIX workstations. It is anticipated that Microsoft NT will also be a platform which will be used for On Line control and monitoring. The increasingly lower price / performance ratio, as well as the continuing expansion of this platform make support for this platform increasingly important.
As the PHENIX detector can be partitioned into separate sub detector granules which can be run independently many of the detector components may be in different operational states. While one granule is taking data, another may be in an initialization state. Since the granules may become part of any independent detector operation, the On Line software must be able to assign and allocate ownership of detector granules in order to avoid conflicts in detector control. In order design a system with such dynamic behavior, the On Line group has sought to develop the On Line system with the aid and advantages of object oriented tools, techniques and methodology where ever possible. C++ has been adopted as the standard language for the majority of the On Line code development In addition, emerging or established industry standards in communication and process control have been adopted as much as possible. Following this philosophy , the On Line group has adopted the Shlaer / Mellor Object Oriented Analysis and Design methodology for the design of the On Line computing system. In addition, CORBA has been adopted as the standard for development of the distributed architecture of the On Line system. The slow controls of the PHENIX detector will be controlled through the EPICS computer system.

3 Online Development methodology

The goals of the Shlaer / Mellor Object Oriented Analysis is to identify the entities or objects which make up the system under analysis. For each identified object in the system, the objects attributes, and relationships to other objects are identified and placed into an information model. For each active object, that is, for each object that has a pattern of behavior over time, a set of active states that the object can exist in are identified. Along with the states, the events that cause the object to enter a given state and actions that occur in that state are also identified. Events are control signals which contain information identifying the target object which is to receive the event, and any data which may be associated with the event. An object that receives an event will use the identifier of the event to cause a state transition to take place in its state machine. Consequently, any actions which are defined for that state are executed. This information is gathered into a state model for the object. Interactions between objects which occur in an objects state are gathered on an object communication model. Finally the processing and the data accessed in an active objects state actions is gathered into an object's process model. The outcome of this analysis is a description of the behavior of all the active objects in the system, and how they interact with each other and with the data in the system.

4 Architecture Description

For the support of the distributed OnLine architecture, Orbix by IONA Technologies, was chosed as the implementation of CORBA. Orbix was chosen as this product implements the full C++ IDL mapping as specified by the OMG, is CORBA 2.0 compliant, is a clear market leader in the supplier of CORBA technology, and has implementations on all the platforms and operating systems which PHENIX currently or intends to support. These operating systems and platforms includes Solaris on a Sparcstation , IRIX on the SGI, VxWorks which is resident in VME based 680X0 processors and Pentium based NT operating systems.
Orbix provides a set of functionality which aids in the development of a distributed architecture. Orbix provides transparent remote object location and invocation services. Object and process filters are available which can provide a mechanism for implementing security access to system objects. Loader service is available to locate an instantiate objects which have been invoked but are not currently available. This service provides support for the implementation of some degree of object persistence.
The main run control operation of the OnLine system is controlled through an application called PHLOPS. This application has been developed entirely

from object models which have been entered into a CASE tool from Project Technologies. The application code is generated entirely from a code generator which was purchased from Project Technology. The code generator accepts SQL statements which are output from the CASE tool and generates the C++ code.
The run control application is responsible for accepting detector configuration data and operator commands for the initialization, configuration, and operation of the PHENIX detector. The run control application contains objects which represents each of the configurable entities in a PHENIX detector partition. These objects are maintained in lists internal to the run control application, which have been generated from the code generator. The objects representing the controllable detector components each encapsulate a state model which the objects transition through. The transition between operational states are triggered by operator commands and response events from hardware components.
The run control program has been interfaced to a Motif based GUI application which presents the interface to the detector operator. This application can accept commands to place components of the data acquisition system into one of their operational states. When the user interface program sends a command to the run control application, an event will be generated and sent to a remote object which provides the implementation of and directly controls the detector hardware components.
The routing and delivery of events throughout the OnLine system is controlled through a mechanism called the Event Notifier. The Event Notifier is a CORBA compliant server that receives requests for and delivers events asynchronously to a destination object In addition to its routing and event delivery functions, the Event Notifier serves to decouple the event supplier from the event receiver. The event generator does not need any knowledge of the location or implementation of the object receiving the event. Thus the client application code is not linked to any object receiver implementation code. The Event Notifier also provides for event filtering and buffering of events.
The Event Notifier works by first having an object that has an interest in receiving an event to be registered with the event Notifier. Registering an object to receive an event results in the creation of a mapping between the object character name and the CORBA object reference. This mapping is maintained by the Event Notifier. Once an object is registered, the event Notifier can look up, and associate its character name identifier with the reference to the object. When an object is registered with the Event Notifier, an event list is submitted with the request for registration. The event Notifier will use this event list to filter events, and to only pass along those events for which the object wishes to be notifed of. Applications use the Event Notifier by first obtaining a reference to it from the ONCS Name Service which will be described below. Once a reference to the Event Notifier has been obtained, a client application can send a command to any registerd object by invoking the sendevent method of the Event Notifier. The sendevent method takes as an argument a structure

which contains the identity of the destination object, an event identifier and any auxilliary data which is required by the destination object for this command.
The Event Notifier has public methods to register, and remove an object as well as to send events to any registered objects. These public methods are defined in the Interface Definition Language (IDL) file for the Event Notifier and is given below.

```
interface EventNotifier {
    typedef sequence<unsigned long> eventIdlist;
    // subscribe an object to receive events in the eventidlist
    oneway void subscribe( in eventIdlist eventidlist, in string destobjname,
  in Object eventreceiver );
  // remove an object from receiving events
  oneway void unsubscribe( in Object eventreceiver );
  // send an event to a registered object
  oneway void sendevent ( in Event event );
};
```

The Event Notifier is modeled after the OMG CORBAservices Event Services. This is a service which is defined by the OMG to provide indirect communication between event producers and consumers. This service employs a construct called an event channel. The event channel appears to be an object which implements either a push or pull model of events. That is, event generators can push events into the event channel and events are pushed on to the event receiver. Alternatively, an event receiver can request or pull events from the event channel and the event channel will in turn pull any pending events from the event supplier. The ONCS Event Notifier mechanism is patterned after the push event model.
The On-Line software is a name based system. Components and services are accessed in the ONCS system by issuing commands to components which are accessed through a character name string. A naming service provides access to ONCS services by associating a character string name with an object reference. ONCS system software will register with the name service, an object reference and the character name by which this object is known to the ONCS system. Client applications can retrieve the object reference from the name service which can then be used to invoke an ONCS service. The name service itself resides at a well known location. It is the key to gaining access to ONCS services. The name service has methods for registering objects with the name service as well as for resolving an object reference when given the character name string for an object. The interface and the public methods of the name service is defined in the Interface Definition Language (IDL) file.
To use the name service a client first binds to the name service server using the Orbix supplied bind function. The bind function returns a reference to the name server. Then the name service method ResolveName is invoked to

obtain a reference to a registered remote service. In the example given below, a reference to the Event Notifier service which has the name identifier of "eventnotifier" is obtained from the name server. The Event Notifier will have been registered with the name server when the server first starts up. Once this object reference is obtained, methods of this remote object can be invoked directly. An example of using the name service to obtain an object reference is given below.

```
// declare and bind to the ONCS name server
nameserv_ptr          pnameservice;
pnameservice = NameServer::_bind(":NameServ"," ");
//Obtain a reference to the remote object "eventnotifier" from the name server
CORBA:: Object_ptr pobject;
EventNotifier_ptr pevnot;
// resolve the remote object name from the name server
pnameservice→ResolveName("eventnotifier",pobject) ;
pevnot = EventNotifier::_narrow(pobject);
// invoke a method on the remote object obtained from the name server
pevnot→sendevent(pevent);
```

5 Active Objects

Events are generated and received in the ONCS software system by active objects. Active objects are those objects which have a pattern of behavior in which actions or functions are executed within well defined states. Objects which have execute such functions are implemented with a finite state machine. Each of these objects will receive an event by executing a public takeevent method which all active objects inherit from a component base class. This method will accept an event and cause a state transition to the object's next state based on the identifier of the received event.
Objects in the ONCS software system are managed by an object called the object manager. The function of the object manager is to manage the creation, activation and removal of all objects that are known to a server. The object manager implements methods which will allow it to dynamically create or destroy objects in its local server. The object manager itself is a CORBA compliant object that is created in a server which is known to the ONCS Name Service. Once objects are created and maintained by the object manager, they become available to receive events from any client in the system via the Event Notifier. The object manager may also return a reference of any object known to the object manager to a requesting client application. This will allow an object's methods to be invoked directly by remote clients if desired.

6 Status

A prototype version of the ONCS architecture has been developed and is currently under test. Test beam runs which exercise components of the PHENIX data acquisition system running under the ONCS system software are scheduled for the Fall of 97.