DELPHI Collaboration

DELPHI 92-98 DAS 127
8 July, 1992

THE DELPHI EXPERIMENT CONTROL SYSTEM

T. Adye[2], L. Beneteau[1], T. Camporesi[1], Ph. Charpentier[1],
M. Dönszelmann[1], B. Franek[2], C. Gaspar[1], Ph. Gavillet[1],
F. J. Harris[3], B. Jones[1], M. Jonker[1], J. G. Loken[3],
P. Moreau[1], E. Murzeau[1], R. Sekulin[2], G.R. Smith[2],
P. Vande Vyvre[1], A. Vascotto[1]

Abstract

The DELPHI detector, which is in operation since the start of LEP in August 1989, consists of 16 different sub-detectors. To provide a high degree of independence to the individual sub-systems, the data acquisition and control system has been split into autonomous partitions. The software organization of these partitions has been standardized up to a high degree. So called equipment computers (EC) provide a fully independent environment to these sub-systems when they are running in stand-alone mode, allowing the detectors to calibrate and test their equipment without interference with other partitions. To cope with the complexity of the experiment control we have developed a new concept for the coding of the control logic.

Paper presented at the 2nd International Workshop on
"Software Engineering, Artificial Intelligence and Expert Systems in High Energy and Nuclear Physics" which took place from 13-18 January 1992 at France Telecom's Agelonde site, at La Londe des Maures, Provence - France

[1] CERN, Geneva, Switzerland
[2] R. A. L., Chilton, UK
[3] Nucl. Phys. Lab., Univ. of Oxford, UK

THE DELPHI EXPERIMENT CONTROL SYSTEM.

T.Adye[2], L.Beneteau[1], T.Camporesi[1], P.Charpentier[1],
M.Dönszelmann[1], B.Franek[2], C.Gaspar[1], P.Gavillet[1],
F.J.Harris[3], B.Jones[1], M.Jonker[1], J.G.Loken[3],
P.Moreau[1], E.Murzeau[1], R.Sekulin[2], G.R.Smith[2],
P.Vande Vyvre[1], A.Vascotto[1].

[1]CERN, Geneva, Switzerland.
[2]R. A. L., Chilton, UK.
[3]Nucl. Phys. Lab., Univ. of Oxford, UK.

Abstract

The DELPHI detector, which is in operation since the start of LEP in August 1989, consists of 16 different sub-detectors. To provide a high degree of independence to the individual sub-systems, the data acquisition and control system has been split into autonomous partitions. The software organization of these partitions has been standardized up to a high degree. So called equipment computers (EC) provide a fully independent environment to these sub-systems when they are running in stand-alone mode, allowing the detectors to calibrate and test their equipment without interference with other partitions. To cope with the complexity of the experiment control we have developed a new concept for the coding of the control logic.

1. INTRODUCTION

The DELPHI detector [1], a general purpose detector with special emphasis on particle identification, is constituted out of 16 different sub-detectors, which were built by different teams of laboratories of the DELPHI collaboration.

To provide a high degree of independence to the individual sub-detectors, the data acquisition and control of the experiment is split into a set of autonomous partitions. Although each detector has its own specific characteristics and function in the particle identification process, the organization of these partitions has been standardized up to a high level. This standardization is extended towards the central readout and trigger. The architecture of these partitions closely resembles an individual detector partition.

The control of all devices associated with each partition, is organized into so called control domains. There are separate control domains to handle the the two aspects of each sub-detector:

- **Data-aquisition:** i.e. the fastbus embedded readout system [2], and the data flow handling in the equipment computers [3],

- **Slow-control:** i.e. the monitor and control system for technical aspects of the sub-detectors [1] such as gas, volts, pressure, temperature, ...

The DELPHI control systems consists of 18 domains for data acquisition, 12 domains for slow control and a few central control domains. The control domains and their associated driver processes are distributed over 20 different nodes in the DELPHI online VAX cluster.

2. THE STATE MANAGER CONCEPT

Historically, high energy physics experiments were almost never complex enough to devise dedicated tools to implement the experiment control logic. From the ancient 'all in one task' approach, and the following foreground / background approach, the distributed data acquisition emerged. Run control logic, taking care of synchronization, resource management, authorization, and command sequencing was spread out over the distributed processes, sometimes assisted by embedded logic in a interactive control process. The logic, often grown *ad hoc*, was coded in FORTRAN and used unsafe mechanisms to cope with mutually exclusive conditions.

The DELPHI experiment control system is characterized by a highly decentralized organization: each embedded processor, the equipment computers, and all the tasks running in these computers have a large extent of autonomy. This organization makes the individual components very flexible and maintainable due to their independence. On the other hand it however puts additional strains on the control system.

To cope with the complexity of the control we developed a new concept for the coding of the control logic [4]. This concept was developed with the following design requirements in mind:

- The control system should be able to deal with the distributed nature of the 'driver processes' (i.e. the processes controlling the external devices), running in the experiment.
- The control system should be easy to modify without major effort (such as the relinking or restarting of unrelated driver processes).
- The control system should operate independently from the driver processes and from any interactive run control process. More specifically, there should be no control logic built into any of these processes. The driver processes should have no knowledge about other driver processes. Likewise, the run control user interface program should have no built in knowledge about the interaction between the different devices.
- The control system should be able to take automatic actions upon changing conditions in the experiment, independent of any operator and/or interactive run control interface process.
- The control system should provide concurrent external access to a control domain. This is to allow central control domains and a local user interface to access concurrently a local control domain. Of course, it is up to the encoded control logic to arbitrate between possible conflicting requests.

The approach we adopted is based on the *State Manager* concept. In this concept, the experiment is described in terms of *objects*, i.e. logical subsystems, for each of which a number of *states* are defined. An object may correspond directly to a concrete entity in the experiment (a computer controlled device) or any abstraction used in describing the experiment provided it can be identified by a 'noun' (e.g. *'run'*, *'trigger'*,*'central detector'*, etc.).

```
! -- Example of SML code

object:   LOCAL_CONTROL
  state:    READY
    action:  BEGIN
      do MOUNT TAPE
      if TAPE not in_state REAY then
        do REPORT_MOUNT_ERROR ERROR_OBJECT
        terminate_action /state=READY
      endif
      do LES BEGIN
      if LES in_state PAUSED and
        RUN_MODE in_state AUTO_START then
        do LES CONTINUE
        terminate_action /state=RUNNING
    . . .
  state:   RUNNING
    when LOG_MODE in_state ACTIVE and DATA_LOGGER in_state FILE_FULL      do PAUSE_RUN
    when LOG_MODE in_state ACTIVE and DATA_LOGGER not in_state FILE_OPEN do ABORT_RUN
    when LES in_state ERROR do ABORT_RUN
    action:  ABORT_RUN
    . . .
object:   LES /associated
  state:    READY
    action:    BEGIN
    action:    INITIALIZE
    . . .
  state:    RUNNING
    action:    PAUSE
    . . .
. . .
```

Example 1: Example of SML code

The control system, which is given by the interaction between the various objects, is specified using a formal language, the *State Manager Language* (SML). The main characteristic of this language are:

- *Finite state logic.* Objects behave as finite state machines. The only 'variables' in this language are the states of the objects (e.g. RUNNING, PAUSED, ...). An *action* (method) applied on an object can bring about a change in its state.
- *Instruction sequencing.* Actions on an *abstract object* may specify a sequence of *instructions*, mainly consisting of actions and logical tests on other objects. Actions on *concrete objects*, are send off as messages to an *associated driver process* which controls the external device.
- *Asynchronous execution.* Several actions may proceed in parallel: an action applied by object-A on object-B, does not suspend the instruction sequence of object-A. Only a test by object-A on the state of object-B suspends the instruction sequence of object-A, if object-B is still *in transition*.
- *AI like rules.* Each object can specify logical conditions based on the state of other objects, which, when satisfied, will trigger an action of the local object. This pro-

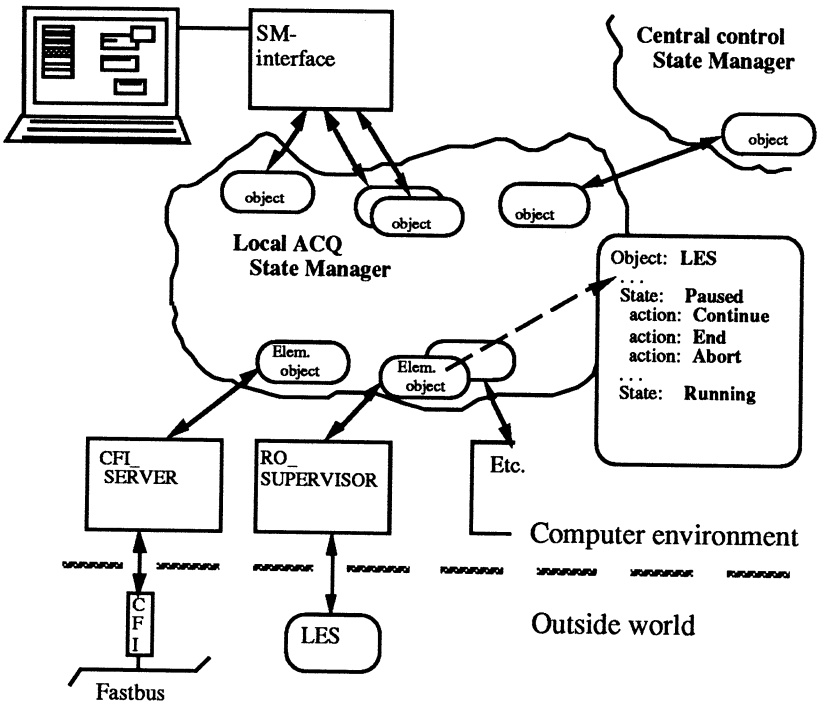

Figure 1: (DELPHI local data acquisition control.

vides the mechanism for an object to respond to unsolicited state changes of its environment.

An example of SML code is given in example 1.

The logic, specified in the State Manager language is translated by a special compiler to create the State manager. The State Manager runs as an independent process and communicates with is environment using DECNET.

An interactive control program can be used to control the state manager. This program has access to the specification and states of the objects and may uses this information to dynamically configure its interface.

3. DELPHI EXPERIMENT CONTROL DOMAINS

To organize the control system of the DELPHI experiment, objects belonging to a specific aspect of the experiment are grouped into an independent *State Manager domain* (SM-domain). All objects in one SM-domain are managed by their State Manager Process. To coordinate the activities of these individual SM-domains, certain 'abstract' objects of these domains have been made visible as 'concrete' objects in the central domains. The following sub sections describe the different type of SM-domains participating in the DELPHI experiment control.

3.1 Local data acquisition domains

Each data acquisition partition is controlled by one State Manager (see figure 1). Because of the high degree of standardization, it was possible to write a single package of SML code for all the 18 partitions. It consist of 12 objects having 2 - 6 states with 3 - 4 actions / state. The total number of SML instructions is 1770. There are two main objects in these domains to orchestrate the running of the partition. The OPERATOR object accepts the top level commands (e.g. *start_run*) from the local operator user interface. The LC object accepts the commands from the central control when the partition operates as part of the whole DELPHI detector.

The processes which are controlled by the local data acquisition domains are the Fastbus Supervisor, the Readout supervisor, the Data logger and an optional Calibration controller.

3.2 Local slow-control domains

The slow-control domain handles the monitoring and control of technical aspects of a sub-detector such as gas and volts. The most important aspect of the slow-control domain, is the control of the High Voltage of the gaseous particle detectors. The raising and lowering of these volts have to be coordinated with the status of the LEP accelerator.

Unlike the local data acquisition domains, there is no full standardization in the local slow-control domains. This is mainly due to the differences in the technical aspects of the individual sub-detectors, such as the requirements on High Voltage control. However, seen from the central control, the individual slow control domains are identical, i.e. they all have a set of identical 'top' objects with the same states and actions. Internally these domains are tailored to the environment required by the operation of the specific sub-detector.

3.3 Central domains

The central control logic brings the individual partitions domains together in a coherent system. By controlling the partition SM-domains, (i.e. the data acquisition and slow control domains), it prepares and supervises the DELPHI detector for global running (see figure 2). There are at present two domains which coordinates the experiment as a whole.

The *central data acquisition domain* controls all the detector data acquisition domains participating in the central run, as well as the central readout control supervisor and the central data logger process.

The *central slow control domain* integrates the partition slow control domains and in particular will provide in the future the interlock between the High Voltage of the detectors with the status of the LEP machine mode.

An important aspect we will introduce in the central control during the coming run, is the status of the LEP machine. This will allow the central control to take automatic actions triggered by the change of state in LEP machine, such as the automatic

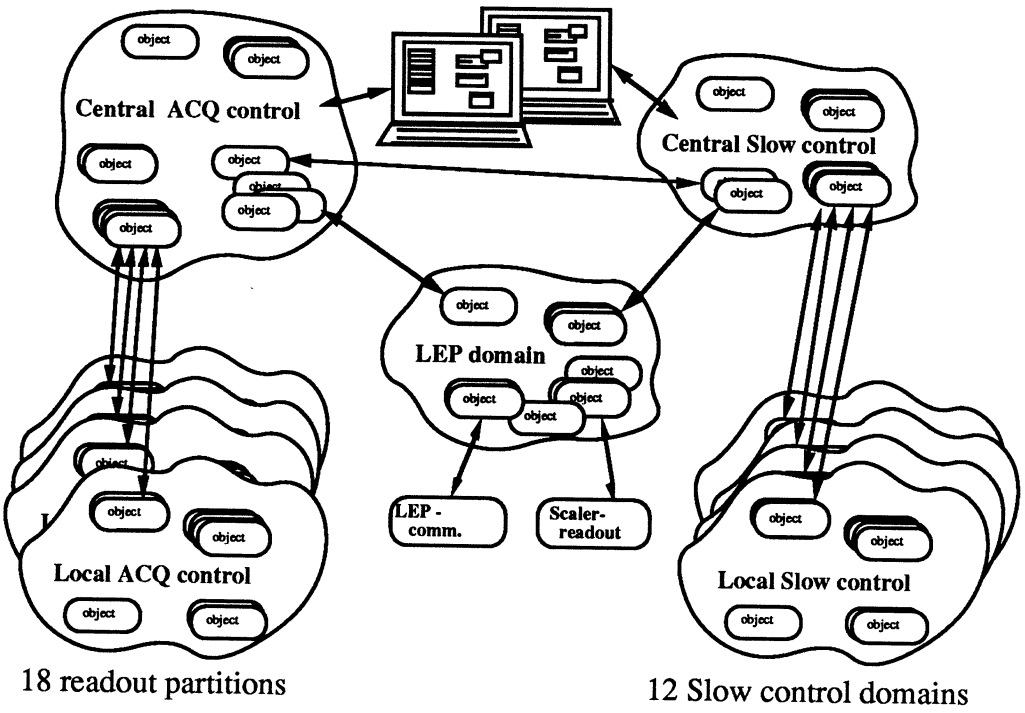

Figure 2: (DELPHI central experiment control.

end of run and ramp down of detector high voltages before LEP prepares a new fill, or the ramp up of detector high voltage and start of run when LEP has finished the preparation of a new fill and the beam conditions are stable.

4. DISCUSSION OF THE STATE MANAGER CONCEPT

The first version of the DELPHI experiment control, based on the State Manager concept, was put into operation in spring 1990. Its inherent modularity made it possible to commission it on a few partitions first. Since then it gradualy matured to the complex system described here. The system has greatly simplified the operator's task of running the experiment and significantly reduced errors. Since the experiment control logic can be changed independently of the driver processes, it has become possible to stage the introduction of new operation procedures.

In the following subsections we try to give an objective assessment of the State Manager concept and its implementation, and we discuss several aspects in which it might be improved.

4.1 Language extentions

The SML language has a restricted set of language elements to manipulate individual objects. This limitation was most apparent, in the central control domains of the experiment, where large groups of almost identical objects are manipulated. A

trivial condition such as: 'When any of the detector partitions which are participating in the central readout, is in the state error do ...' is not simple to code.

To avoid writing many lines of almost identical code for testing or acting on large sets of objects, we have added an SML preprocessor. The macro expansion of the above example generates roughly 100 lines of SML code to control the 18 partitions of Delphi. In the following items we suggest how the existing language could be extended in order to handle such problems more naturally.

- *Dynamic object sets* Objects of the same type (*class*) could be member of a so called *dynamic object sets*. Special statements should allow to add or remove dynamically objects from these object sets. Actions and tests performed on an object set are performed on all current members of the object set. In the case of tests, one has to further specify how the results on the individual members should be combined: e.g. `when` *any*`(global_detector) in_state error do ...`, or `if` *all*`(global_detector) in_state ready then ...`.
- *Dynamic object creation* In certain cases the system contains a number of object which is *a priory* unknown, and which could change during the running of the system, depending on configuration parameters. (e.g. the number of secondary data loggers). To handle these situations would require the dynamic creation of new objects of a given class. The control logic would further control these dynamicly created objects through object sets.
- *Object properties* Beside the state of the objects, the SML language does not handle any other kind of variables. This could be improved by extending an existing feature of the language that has not been mentioned so far: i.e. *object properties*. Object properties are predefined properties that can be attributed to objects, e.g. the property 'operator_display' could be assigned to all objects which should be displayed by the user interface. Moreover, object properties can have values assigned and hence are typed: either fixed-keyword, integer, or string. At present the values of object properties are static, they are defined at compilation time and cannot be changed subsequently.

  By making the property values dynamic and allowing some simple operations on properties (e.g. tests, using their values as command parameters, ...) a very powerful concept can be added to the language.
- *Command arguments* The present State Manager implementation has the possibility to add parameters to the commands which are send to the driver processes. These parameter values can be either 'hard coded' in the language, or they can be determined at run time by a logical name translation. Although, this later option largely extends the flexibility of the State Manager concept, at present there is no foreseen way to define the logical name at run time.

  On the other hand, the mechanism to pass command parameters by logical name, could lead to the introduction of new objects (i.e. logical names) in a control domain with hidden states (the logical name values). One can eliminate this risks by allowing the parameter value to be taken from the state of an internal abstract object.

4.2 Networking

The performance of the system is largely dependent on the communication efficiency between the various components of the control system, i.e. the exchange of information on states and actions between the State Manager processes and the associated driver processes.

In the present implementation there is one central transaction dispatcher for all processes on the whole cluster involved in experiment control. This transaction dispatcher is even used when information has to be exchanged between two processes on the same CPU, hence adding an unnecessary load to the network system. Since this central transaction dispatcher, has to handle many logical links, (in the Delphi experiment, the number of links is approaching the number 200) the dispatching can even form a severe bottleneck at 'peak demand' times. These peak demand times typically happen when the whole detector is being prepared for global running conditions, a situation where fast response time would be most appreciated.

Moreover, due to the usage of the central network transaction dispatcher, we also found that it was inefficient to encode 'device internal logic' in SML. This logic, which deals with the internal states of a controlled device, and which does not interact with the overall control logic, is now mostly hard coded in the driver processes rather than in SML.

A solution we are presently studying [5] is to integrate the transaction dispatching function into each individual process which participates in the state manager domains. In this case, there is still a central name server required to register the network coordinates of the 'information access points'.

5. CONCLUSIONS

The DELPHI experiment control, running since 1989, is based on the State Manager concept. This concept uses a dedicated object oriented language, with some very interesting built in features such as asynchronous execution and simple AI like rules.

With this tool we have transformed the initialization and setup of the data acquisition system from a sequence of complex operations into a simple operator action which completes in short time. The State Manager based experiment control has now been in use since 2 years for the data acquisition domains and since one year for the slow control domains. In the near future we will enhance the experiment control even further and include facilities for automatic error recovery, and automatic run control based on the status of the LEP machine.

Although the experiment control system handles quite well the run control operation, it is not suited to handle run configuration setup or detailed run status information. For this purpose we are presently adding new interactive tools based on MOTIF [5]. We will also improve some of the other shortcomings we have experienced in the concept and in its implementation.

REFERENCES

[1] DELPHI collaboration, P.Aarnio et al., The DELPHI Detector at LEP, *Nucl. Instr. Meth.* **A303** (1991) 233-276.

[2] DELPHI DAS group, W.Adams et al., The DELPHI Fastbus Readout System, Presented at the CHEP-91 conference, Tsukuba, March 1991,

[3] DELPHI DAS group, T.Adye et al., Architecture and Performance of the DELPHI Data Acquisition and Control System, Presented at the CHEP-91 conference, Tsukuba, March 1991,

[4] J.Barlow et al., Run Control in MODEL: The State Manager, IEEE trans. nucl. sci. 36 (1989) 1549-1553

[5] DELPHI DAS group, C.Gaspar et al., DELPHI Experiment Control Integration, These proceedings.