A HIGHLY DISTRIBUTED CONTROL SYSTEM FOR A LARGE SCALE EXPERIMENT

C. Gaspar

CERN, European Organization for Nuclear Research
CH-1211 Geneva 23, Swizerland

J. J. Schwarz

INSA L3I. B502 IF
20 av A. Einstein
69621 Villeurbanne Cedex, France

Abstract These days physics experiments can no longer be acomplished by a single "genious" in his laboratory, they involve large international collaborations of hundreds of scientists, enourmous particle accelerators and very complex particle detectors.
This paper will give an overview of the problems encountered and the solutions retained when building control systems for supercolliders, the largest scientific instruments ever built by humans.
Keywords Computer communication, Control Systems, Distributed control, Reliability, Robustness

INTRODUCTION

DELPHI (DEtector with Lepton, Photon and Hadron Identification) (DELPHI Collaboration, 1991) is one of the four experiments built for the LEP (Large Electron-Positron) collider at CERN, the European Organization for Particle Physics.

DELPHI consists of a central cylindrical section and two end-caps. The overall length and the diameter are over 10 meters and the total weight is 2500 tons.

The electron-positron collisions take place inside the vacuum pipe in the centre of DELPHI and the products of the annihilations fly radially outwards. The products of the annihilations are "tracked" by several layers of detectors and read out via some 200,000 electronic channels. A typical event requires about 1 million bits of information.

The DELPHI detector is composed of 20 sub-detectors, as described in Fig.1, which were built by different teams of laboratories of the DELPHI collaboration (around 800 scientists from 50 laboratories all over the world).

The main aim of the experiment is the verification of the theory known as the "Standard Model".

The DELPHI experiment started collecting data in 1989 and it has to be up and running 8 months/year (24h a day) until around the year 2000.

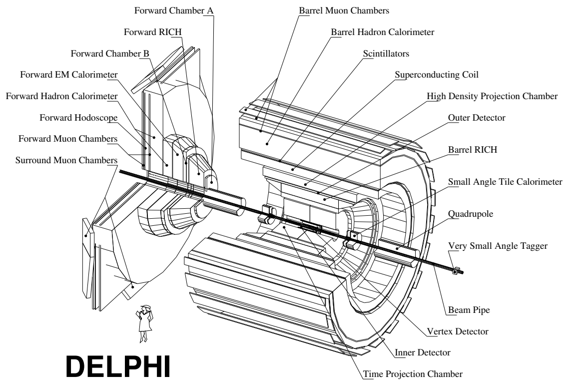

Fig. 1. The DELPHI Detector

During its life time the experiment is constantly modified, to allow for different physics studies, new sub-detectors can be introduced and old ones can be upgraded or replaced.

The control system of the experiment has to assure that the experiment works efficiently and reliably during the running periods and it has to allow for an easy reconfiguration of any part of the experiment according to the physicists wishes.

DELPHI ONLINE SYSTEM

The online system of a physics experiment is composed of many different parts, its main tasks are:

- The Data Acquisition System (DAS) (Charpentier et al., 1991)

  Reads event data from the 20 sub-detectors composing DELPHI and writes them onto tape. In order to provide a high degree of independence to the individual sub-detectors the DAS system has been split into 20 autonomous partitions. These partitions are normally combined to form a full detector but they can also work in stand-alone mode for test and calibration purposes.

- The Trigger System (Fuster et al., 1992)

  Provides the DAS system with the information on whether or not the event is interesting and should be written to tape. The final trigger decision is a combination of the partial decisions of the sub-detectors.

- The Slow Controls System (SC) (Adye et al., 1992)

  Controls and monitors slowly moving technical parameters and settings, like temperatures, pressures and high voltages of each sub-detector, and writes them into a database.

- The Lep Communication System (Dönszelman and Gaspar, 1994)

  Controls the exchange of data between the LEP control system and DELPHI.

- The Quality Checking System (QC)

  Provides automatic and human interfaced tools for checking the quality of the data being written on tape.

The complexity of controlling such a system comes from the fact that although the different parts of the system have different requirements and constraints, ranging from real time behaviour in the DAS system to strict safety constraints in the SC area they have to work together for the common goal of providing "good" data for physics analysis.

In previous experiments the control of the different areas was normally designed separately by different experts, using different methodologies and tools resulting on a set of dedicated control systems.

DELPHI decided to take a common approach to the full "experiment control" system. The result was the design of a system that can be used for the control and monitoring of all parts of the experiment, and consequently obtaining a system that is easier to operate, because it is homogeneous, and easier to maintain and upgrade.

The Online control system is characterized by a highly distributed architecture, as most current computer control systems, it consists of workstations interconnected by a local area network. Each workstation (through a Graphical User Interface - GUI) controls and monitors a part of the system, either a sub-detector (Det) or a central task, like DAS or SC as shown in the diagram of Fig. 2.

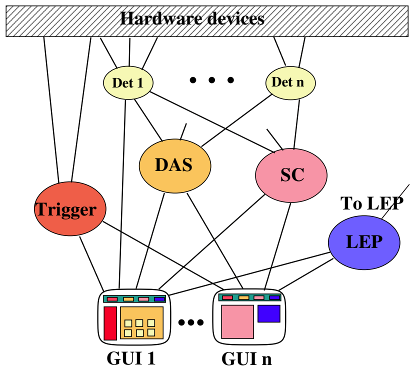

Fig. 2. The Online System

The need for the multiple tasks composing the DELPHI control application to run on different machines brought up the problem of communicating easily, effectively and reliably among processes and processors.

In order to solve this problem DELPHI has designed it's own communication system - DIM (Distributed Information Management System) (Gaspar and Dönszelman, 1993).

DESIGN REQUIREMENTS

The control system of the experiment is composed of more than 500 processes distributed over around 40 workstations.

As in most distributed systems, DELPHI had to face critical design issues like :

- locating tasks and data resources distributed across the network,
- establish and maintain interprogram communication on the network,
- coordinating the execution of distributed tasks,
- synchronizing replicated programs or data to maintain a consistent state,
- detecting and recovering from failures in an orderly, predictable manner.

The purpose of the DIM system is to implement a coordination model able to handle these diverse issues coherently.

DELPHI adds some aditional requirements:

- An efficient communication mechanism

  DELPHI has some requirements on what concerns the communication mechanism, this issue will be discussed in the next chapter.

- Uniformity

  The DIM system should be capable of handling all process interactions withing the online system, all processes involved with control, monitoring, processing or display should use the same communication system. An homogeneous system is much easier to program, to maintain and to upgrade.

- Transparency

  An important goal for a distributed communication system is tranparency.

  At run time no matter where a process runs, it should be able to communicate with any other process in the system independently of where the processes are located. Processes should be able to move freely from one machine to another and all communications should be automatically reestablished (this feature also allows for machine load balancing).

  At coding time the user should not be concerned with machine boundaries, the communication system should provide a location- - transparent interface.

- Wide-area availability

  DELPHI is an international collaboration, any necessary information should be available to the outside world, using the same system.

By fullfilling such requirements a communication system can greatly improve the performance of the complete system. It provides a decoupling layer between software modules, that makes coding, maintenance and upgrade of the system easier and improves efficiency and reliability at running time.

COMMUNICATION MECHANISM CONSIDERATIONS

When designing a distributed control system, the choice of the communication mechanism to be used is an important issue.

Distributed applications are often based on Remote Procedure Calls (RPC) (Birrell and Nelson, 1984). In the RPC mechanism the client sends a message containing the name of a routine to be executed and its parameters to a server, the server executes the routine and sends a message back containing the result. This implies that the communication is point-to-point and synchronous since the client always waits until the routine finishes execution.

For an application like DELPHI the RPC mechanism is very heavy and not well suited. In DELPHI's Online system some processes have to react to condition changes and many times multiple processes have to be notified of these changes.

A more suited model for this type of requirements is one that allows for asynchronous and one-to-many (group communications) (Kaashoek and Tanenbaum, 1991) communications.

The solution we thought the best in our case is for clients to declare interest in a service provided by a server only once (at startup), and get updates at regular time intervals or when the conditions change.

This mechanism - interrupt like - as opposed to RPC's polling approach involves twice less messages sent over the network, i.e. is faster and saves in network bandwidth. It has also the advantages of allowing parallelism (since the client does not have to wait for the server reply and so can be busy with other tasks) and of allowing multiple clients to receive updates in parallel.

This approach together with the possibility of sending commands to servers (more RPC like) are the main features of the DIM Communication Mechanism.

DESIGN PHILOSOPHY

DIM, as most communication systems, is based on the Client/Server paradigm.

The basic concept in the DIM approach is the concept of "Service". Servers provide "Services" to Clients. A service is normally a set of data (of any type or size) and is recognized by a name - "Named Services". The name space for services is free.

In order to allow for the required transparency (i.e a client does not need to know where a server is running) as well as to allow an easy recovery from crashes and migration of servers a Name Server was introduced.

The architecture developed for the interactions between servers, clients and the name server in order to fullfill the previously mentioned requirements is the following :

- Servers

  To become a DIM server, a process has to accomplish the following operations:

  ⋄ Register the Services, by calling the routine *dis_add_service* for each service it wants to publish.

The service can be provided in two ways, either by describing the address and the size of the data to be passed to the client or by specifying a routine that will prepare the data and return its address and size.

⋄ Register commands (if any) expected from clients, by calling the routine *dis_add_cmnd* specifying a routine to be executed on command arrival.

⋄ Start serving client requests. By calling *dis_start_serving* the list of provided services will be sent to the name server and clients will start beeing served.

From then on whenever a client service request arrives, its parameters are stored and the service will be sent to the client whenever necessary, according to the requested update mechanism. The routine *dis_update_service* can be used to force the dispatching of the service to the clients (that have enabled this type of update).

Example :
A program publishing the current state of the experiment can call *dis_add_service* specifying "DELPHI/STATE" as service name, the address and size of a character string buffer where the current state will be stored and NIL as routine address.

service_id =
dis_add_service("DELPHI/STATE", state_buf, state_size, 0);
dis_start_serving("DELPHI_STATE");

In the main program, possibly by using as a client many different DIM services, it will compute the overall state of the experiment store it in *state_buf* and call *dis_update_service(service_id).*

- Clients

  Any process can access a service by using the routine *dic_info_service.*

  The client can get the service by specifying a buffer address and size where the service data will be stored and/or by specifying a callback routine to be executed on service arrival. In fact when using the buffer address mechanism the system works as if the clients maintain in cache a copy of the server's data (the cache coherence being assured by the server).

  The client can also specify the address and size of a constant to be copied to the service buffer and/or passed to the callback routine whenever the connection to the server breakes or can not be established.

  The update mechanism can be of three different types :

  ⋄ ONCE_ONLY : The service data will be accessible to the client only once. (Very rarely used)

  ⋄ TIMED : The Service data will be updated at regular time intervals (Used normaly for the update of discrete quantities like trigger rates or event sizes)

  ⋄ MONITORED : The Service data will be updated whenever it changes, available only if the server provides it by calling *dis_update_service.* (Used for passing states or error conditions)

  Clients can send commands to servers by using *dic_cmnd_service* specifying the address and size of the command data. Clients can also disconnect from services using *dic_release_service* specifying the service identifier returned by *dic_info_service.*

  TIMED and MONITORED services are only requested once by the client (normally at start-up), the service will then be updated automatically by the server.

  When using MONITORED services the server will update the information of all clients whenever it changes, thus making sure the data is coherent over all the clients of a certain service.

  Example :
  A program displaying the current state of the experiment can call *dic_info_service* specifying "DELPHI/STATE" as service name, MONITORED as update mechanism, the address and size of a character string buffer where the state will be stored by the server, the address of the the callback routine that will update the screen when the state is updated and the string "NO LINK" as constant in case of connection failure.

  dic_info_service("DELPHI/STATE", MONITORED, state_buf, state_size, update_routine, routine_tag, "NO LINK", 8);

  By making this call at program statup, *update_routine* will be called every time the state of DELPHI changes, all it has to do is print the variable *state_buf,* in case the server that provides this service dies "NO LINK" will automatically be printed. If the server restarts the correct value will again be printed. In the meantime the main program can be busy doing anything else

- Name Server

  The Name Server acts as a service directory: it keeps an up-to-date list of all the services and servers available in the system. It handles

Server registration by storing in an hash table (of linked services) the services provided by the server. All servers stay permanently connected to the name server and send alive messages from time to time so that the name server can be sure of their availability (when they die or hang their services are discarded). If a server tries to register a service that already exists the name server sends back a kill signal and the server dies with a clear error message (Services have to be unique).

Whenever a client wants to access a service it first contacts the name server, the name server replies the node and address of the server and its communication protocol. From then on the client contacts the server directly. If the service is not available the client receives a negative answer but the request stays queued in the name server and when/if the service becomes available the client will be warned and the connection established.

If a server dies or hangs while a client is connected to it, the client will recontact the name server, the server might have moved to another machine, anyway the request stays in the name server for whenever and wherever the service is again available.

Both servers and clients have retry mechanisms, if the name server is unreacheable they will keep on retrying at random (within limits) time intervals (to avoid network congestion) and whenever it comes back to life servers will re-register all their services and clients will finally request the services they need. While the name server is down all client-server connections previously established continue working.

The Name server also keeps some statistics about servers and services, these statistics are available as DIM services.

Figure 3 shows a small example of the usage of the DIM system within the DELPHI Online System

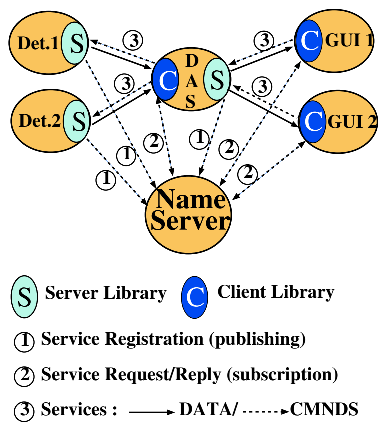

Fig. 3. DIM example

IMPLEMENTATION ISSUES

- Transparency/Easy-to-use

  The complete client and server functionality described above is hidden from the user inside library routines.

  Once a Server has "published" its services or a client has "subscribed" to the services it needs, the handling of client requests or server updates can be done (if desired) without any notification or intervention of the user process.

- Monitoring and Debugging

  The behaviour of complex distributed applications can be very difficult to understand without the help of a dedicated tool.

  The DIM System provides a tool - DID, Distributed Information Display - that allows the visualization of the processes involved in the application as shown if Fig. 4.

  DID provides information on the Servers and Services available in the system at a given moment and on the clients using them.

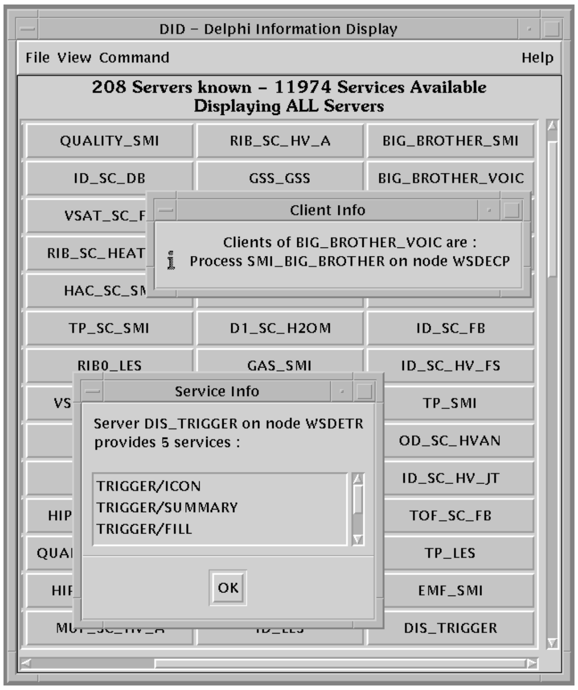

Fig. 4. Display Tool

- World Wide Access

  An application built using DIM can be distributed over the world provided that either TCP/IP or DECNET are available.

  The information available as DIM services can also be accessed by WWW (World Wide Web) through a WWW-gateway. The WWW page can be composed with the help of a dedicated editor.

CURRENT DEVELOPMENTS

Although the DIM system is currently in use in the DELPHI experiment, the project is not finished, its installation on different platforms and over other network protocols would be of great use for both DELPHI and other potential users.

The DIM system is for the moment only available under the VMS operating system and it uses as network support TCP/IP and/or DECNET. The extension to UNIX and OS9 over TCP/IP is beeing studied with the associated problems of different data representations over different machines.

The data formats and the network protocol to be used on each connection will have to be negotiated between the server, the client and the Name Server.

CONCLUSIONS

DELPHI is one of the largest Physics experiments in the world, it's online control system is composed of many different components distributed over many machines. In order to allow for efficient communication among machines and processes a communication system - DIM - was developped.

DIM has greatly simplified the coding and maintenance of the DELPHI online software by providing a network-transparent inter process communication mechanism. The distribution of and the access to up-to-date information of all parts of the system takes place with minimum addition of user code.

DIM's asynchronous communication mechanism allows for task parallelism and multiple destination updates. It's characteristics of efficiency and reliability have considerably improved the performance and robusteness of the complete online system. The number of crashes was reduced from once a week (taking about two or three hours to recover) to none during the last year.

The access to DELPHI information is possible from all over the world either directly through DIM or via WWW through a WWW-DIM gateway.

DIM is responsible for most of the communications inside the DELPHI Online System, in this environment it makes available around 15000 Services provided by 300 Servers. DIM is now also beeing used by other experiments at CERN.

The extension of the DIM system to other platforms would be of great use for DELPHI and is beeing studied.

REFERENCES

DELPHI Collaboration, Aarnio, P. et al. (1991). The DELPHI Detector at LEP. Nuclear Instruments and Methods in Physics Research A303 pp. 233-276.

Charpentier, Ph. et al. (1991). Architecture and Performance of the DELPHI Data Acquisition and Control System. Proceedings of the International Conference on Computing in High Energy Physics '91. Tsukuba, Japan.

Fuster, J.A. et al. (1992). Architecture and Performance of the DELPHI Trigger system. Proceedings of the IEEE 1992 Nuclear Science Symposium. Orlando, Florida.

Adye, T. et al. (1991). The Slow Controls of the DELPHI Experiment at LEP. Proceedings of the International Conference on Computing in High Energy Physics '92. Annecy, France.

Dönszelmann, M. and Gaspar, C. (1994). The DELPHI distributed information system for exchanging LEP machine related information. Nuclear Instruments and Methods in Physics Research A352 pp. 280-282.

Gaspar, C. and Dönszelmann, M. (1993). DIM - A Distributed Information Management System for the DELPHI experiment at CERN. Proceedings of the IEEE Eight Conference REAL TIME '93 on Computer Applications in Nuclear, Particle and Plasma Physics. Vancouver, Canada.

Birrell, A. D. and Nelson, B. J. (1984). Implementing Remote Procedure Call. ACM Trans. Comp. Syst. Vol. 2, No. 1, pp. 39-59.

Kaashoek, M. F. and Tanenbaum, A. S. (1991). Group communication in the Amoeba distributed operating system. Proceedings of the 11th International Conference on Distributed Computing Systems. Arlington.