THE EVOLUTION OF THE DELPHI EXPERIMENT CONTROL SYSTEM OR HOW TO SURVIVE TEN YEARS OF RUNNING

A.Augustinus, Ph.Charpentier,M.Donszelmann, C.Gaspar, Ph.Gavillet and M.Jonker,
CERN, Geneva, Switzerland
T.Adye, B.Franek, R.Sekulin, G.Smith,
RAL, Didcot, England

Abstract

The DELPHI experiment at CERN's LEP collider [1] started taking data in august 1989. During these ten years of running, the DELPHI Experiment Control System (ECS) has been in permanent evolution. However the initial concepts are still present despite upgrades and reengineering. This is mainly due to a careful architecture and design which allowed a smooth evolution with far better functionality but minor changes for the users (programmers and operators)..

1 BASIC CONCEPTS

DELPHI is one of the four large experiments at the LEP electron positron collider. Its readout electronics consists in over 250 000 channels (after multiplexing) and its slow controls system handles several thousands channels (power supplies, temperature and pressure probes…). The Data Acquisition System (DAS) [2] and the Slow Controls System (SC) [3] are the two main components of the Experiment Control System (ECS). Both are organized in a hierarchy of processors and processes (front-end level, sub-detector level and central level).

Besides these two majors system, the ECS integrates the control of the trigger system [4] (two levels of triggering, local and central timing configurations) as well as the LEP Communications System (LCP).

1.1 The Data Acquisition System

The DELPHI Data acquisition System (DAS) [2] is organized as a set of 20 predefined partitions (equivalent to a sub-detector or part of it) and a so-called Central Partition which builds events and performs a software selection of events (Third Level trigger). Each partition is itself split at the front-end level into between 1 and 13 entities controlled each by a Crate Processor (CP).

The standard for front-end electronics and processing is Fastbus. DELPHI has about 150 Fastbus crates.

The partitions are controlled by a Local Event Supervisor (LES) while the Central Partition is under control of the Global Event Supervisor (GES). The CP, LES and GES software are running in the same type of embedded processor, the Fastbus Inter-segment Processor (FIP) [5]. DELPHI uses 72 FIPs running the OS9 operating system and connected to the LAN with TCP-IP.

1.2 The Slow-Controls

Each sub-detector has its own independent Slow-Controls System (SC). The front-end for SC is based on G64 electronics, controlled by M6809 without operating system. The only communications medium with the outside world is Ethernet and softwarewise a CERN-made implementation of RPC over the CATS protocols. The whole system contains about 50 G64 processors.

Another 30 processors are used for controlling the DELPHI gas supply system. This is independent on the sub-detector SC and is based on a standard set of processes but it is formally part of the SC System.

1.3 The Trigger System

The DELPHI Trigger System consists in a decision part and a timing part. Here as well, the organization is hierarchical with a Central and a Local level. The Central decision is performed by a set of Look-Up Tables for each of the two levels of trigger (respectively 3 and 39 μs after each beam crossing), while the Trigger Decision Boxes for each sub-detector are custom made.

The Central Timing is also distributing timing signals to the sub-detectors via standard modules (at least one per partition).

The trigger system also comprises a special readout of a large set of counters for monitoring various parts of the detector (background, trigger rates…). This readout must be decoupled from the standard DAS since it must be permanently running.

1.4 The LEP communications

DELPHI exchanges information with the LEP machine: settings and parameters from LEP to DELPHI, luminosity, magnet status and backgrounds from DELPHI to LEP. This is ensured by a set of processes (LCP) running in the DELPHI environment with the appropriate communication tools with LEP (started with RPC, now using direct read from / write into the LEP database).

1.5 The Online VMS Cluster

DELPHI had chosen VMS as an operating system for its online system, due to its rather good real time response and the flexibility offered by the VMS clusters of grouping as set of machines and communicate between them. A dedicated member of the Online VMS Cluster has been assigned to each sub-detector, while several members of the cluster were dedicated to Central Operations. Supervisors are running on this cluster as well as most User Interface processes.

1.6 The Supervisors

For DAS, each partition is supervised by an LES-supervisor which communicates directly with the embedded LES, while the GES-supervisor controls the Central Partition.

SC entities are supervised by so-called *Elementary Processes* (EP). A standard skeleton for G64 processes [6] as well as for EP's have been developed for most sub-detectors while dedicated software has been developed for the most demanding ones.

1.7 Communications

RPC (over CATS) [8] is used for communications between SC's G64 and the Elementary Processes.

For DAS, since embedded processors are on Fastbus, communications with VMS went initially through dedicated Fastbus modules and VAX interfaces (CFI for local partitions, CHI for Central Partition) and associated software. Hence each partition had a CFI-Server, itself interfaced to other processes with VMS mailboxes. TCP-IP, although available on the FIP's as from 1990, was only used for software loading via NFS (prior to this, RS232 was used for this purpose!).

2 SMI, BACKBONE OF THE ECS

When conceiving the architecture of the DELPHI ECS, it was decided to base it on a single high level system called SMI (State Management Interface).

SMI is based on the concept of *objects* that are in a definite *state* and to which one can send at any moment *an action request* that is meant to bring it into another state. The behavior of objects is described as a Finite State Machine (FSM) through a dedicated language (SML).

Objects are grouped in *SMI Domains* (objects closely related). Two types of objects can be defined :

- Abstract objects: their behavior is fully defined in SML inside an SMI Domain. They may send or receive action requests to/from other objects and their state is determined usually as a result of the change of states of those other objects.
- Associated objects: those represent concrete entities in the system and their behavior in response to actions consists in a concrete interaction with hardware. The content of the action as well as the determination of their states is performed by a dedicated process called *SMI proxy*.

A very simple API has been defined for proxy initialization, the reception of action requests and setting of state changes.

The first implementation of SMI [7] (done in collaboration with CERN's DD/OC group following a DELPHI proposal) was based on a translation of SML into ADA code which then was turned into a specific process for each SMI domain. The SMI Run Time Library (RTL) was as well implemented in ADA.

The communication between SMI processes and proxies was based on the OSP protocol which is highly centralized (all processes send to and receive from a central server).

3 THE INITIAL SYSTEM

When the experiment was commissioned in 1989, each sub-detector was controlled by a MicroVAX II. For those who didn't know that time, these machines had 3 Mbytes of RAM, a 500 Mbytes disk and a CPU power of about 1 MIPS. The state-of-the-art for User Interfaces was based on full screen menus (MHI at CERN) or command line interactive programs. Communications had all to be home made since no real standard existed yet. Hence DELPHI was using CERN developed packages: CATS, RPC or proprietary media (VMS mailboxes and DECNet/T4).

3.1 The early days

At that time SMI was not yet available. Despite that, since its API had already been defined, a temporary version of the SMI-RTL has been set up receiving action requests and setting states via VMS mailboxes.

The DAS User Interface (UI) was very rudimentary and consisted in an interactive process with line command where the operator could see the states of the objects he had to interact with and send actions to them.

For SC as well, the first version of the Elementary Processes (meant to be SMI-proxies, hence later *call SC-proxies*) was also using the mailbox interface, while some sub-detectors were also using interactive G64 programs directly.

No Central synchronization was possible and many operators were needed to send all commands and communicate… by phone or intercom the success (or failure) of their actions. Hence not less than 20 persons were needed to tentatively prepare DELPHI for running and control its normal behavior!

3.2 The infancy

When SMI was made available after the LEP pilot run of summer 1989, it could be used straight on, since the API used for the mailbox interface was that of SMI. Moreover, the use of VMS shared libraries allowed to replace the Mailbox-SMI by the genuine SMI very easily (no recompilation/relink) needed.

The first version of SMI had no other possibility of user interaction but via VMS Global Sections. Hence an SMI domain could only be controlled on the local node where it was running. A generic menu program for surveying the state of SMI objects and send action requests to them has been developed which represented a big step forward for controlling the experiment.

3.3 The Childhood

Another major step forward was made when both Central DAS and SC SMI domains have been completed. They allowed controlling fully each sub-detector from a Central place.

One should say that during all that time, the LEP communication system was still living its own separate life with ad-hoc interactive displays.

Concerning the trigger, from the design phase it had been decided that the sub-detector decision and timing modules would be initialized and supervised by DAS processes (LES and its supervisor). The Central Trigger however was controlled by a dedicated interactive process, in charge of central timing initialization and look-up tables loading. A separate process was in charge of receiving the scalers' contents and make them available for display and recording in a VMS global section.

4 THE DIM REVOLUTION

When more modern User Interfaces became available DELPHI decided to upgrade their full screen menu programs to MOTIF. During the process of design, it was felt as a constraint to have to get information from VMS Global Sections: the UI had to run on the local node. Hence a new project was started with as an aim to provide an efficient distribution of information throughout the online cluster. This is how the *Distributed Information Management (DIM)* package was first introduced in 1993.

DIM [9] is based on the publish/subscribe variant of the server/client paradigm: when a process produces information that may be of some interest for other processes, it publishes it as a *DIM-Service* by registering to a *DIM Name-Server*. Other processes interested by this information may place an inquiry to the Name-Server that will reply with information on the publishing process. The subscriber then directly establishes a link with the publisher which will update information either on a time-interval basis (defined by the subscriber) or when decided by its programmer.

DIM has got a very simple API for publishers (servers) and for subscribers (clients). Hence any process in the Online Cluster can easily become a DIM publisher or subscriber. DIM clients may receive data in a user-supplied buffer or by executing a callback routine. All data transfers in DIM are asynchronous.

DIM takes care of data formats (for inter-platform data exchange). Very important is its ability at recovering from both publishers and subscribers crashes or restarts, without any special action being need from the other processes.

DIM also has the possibility for servers to declare accepted commands and for clients to send commands to servers. Commands consists in sending any data which wakes up the server and executes a callback routine.[1]

4.1 DIM as SMI communications layer

Once DIM had been implemented as a data exchange package in 1993, it was realized that it was also ideal to replace the OSP package as the SMI communication layer. OSP had the great disadvantage of being centralized and hence highly non-scalable, unlike DIM. SMI programs simply publish the states of objects and the allowed actions as DIM services. They receive SMI action requests as DIM commands.

In a first instance, the API of the communication layer of OSP (called ICT) was emulated with DIM. This was very useful since one didn't want to make any modification neither to the ADA generated programs nor to the ADA SMIRTL. Hence only the OSP library was replaced with these mapping routines.

4.2 DIM as a data publisher

DIM is also of course used for its primary goal, i.e. publishing information which can be used by other applications or User Interfaces (as the generic DELPHI User Interface, DUI, for which it had been designed).

As an example, the contents of all scalers read out permanently by the Trigger System are published. They are then used by a large variety of programs, as those computing the background conditions and the luminosity to be sent to the LEP machine.

5 CONTROLS AUTOMATION

Since SMI allows any level of abstraction, it was rapidly realized that one could build on top of the basic controls of DAS and SC more powerful SMI domains to automate and integrate the full ECS.

5.1 The DAS Autopilot

The Central DAS SMI contains already a high level of abstraction, since the operator essentially interacts with one single high level object to which he can send Run Control commands..

It was hence fairly easy to implement an SMI object which would mimic what an operator would do to keep the system taking data, i.e. automatically recover from errors, start a new run when the system is ready etc…

[1] Documentation can be found on WWW at the following URL: http://delonline.cern.ch/dim/doc/www_manual/dim.html

This *AutoPilot* is automatically disabled as soon as the operator takes over by sending a command directly to the Central Run Control (this is mandatory to be able to e.g. stop a run!).

5.2 Full Integration : Big Brother

During data taking, shifters in charge of the DAS and SC have to exchange information in order to start taking data only when the volts have been set onto the detectors. They also have to watch the state of the LEP machine such as to ramp volts when LEP has collisions and background conditions are good, lower the volts at the end of a fill or when background conditions are bad. This may lead to delays or lack of reaction in case the shifters are busy solving another problem.

The trigger system also was left aside of the game and the DAS shifter had to ensure that the proper trigger conditions had been set when starting taking physics data while these conditions might have been changed during the interval between fills.

In order to avoid these delays and potential mistakes, it was decided to fully integrate these two important domains (Trigger and LEP) as SMI domains. They are easily modeled objects with well-defined states and actions.

A very high level SMI domain has been defined which ensures the proper sequencing of operations whenever LEP changes it state. This domain watching everything has been called … *Big Brother!* (BB) [10].

As an example, when LEP is in good conditions for physics data taking, BB stops the current DAS run, sends an action request to SC to ramps the volts, another one to Trigger for setting the proper triggering conditions, as well as for starting logging data. When both SC and Trigger are ready (actually one requires only that the essential sub-detectors are ready), BB sets the DAS AutoPilot on, which then starts a run.

Whenever a fault condition happens in the SC or LEP (e.g. very high background), BB pauses the run temporarily to avoid taking bad data and resumes data taking when SC and/or LEP are back in acceptable conditions.

5.3 SMI reengineering

In order to improve maintainability and portability of SMI, it was decided in 1996 to fully reengineer SMI using OO techniques. Full backward compatibility at the SML level was kept while many new features have been added to the original design. The API was kept unchanged for what concerned the compatible features.

A generic engine is running as a process for each SMI domain. It configures itself at run-time from the SML source code.

The engine as well as all accompanying tools have been written in C++ and this SMI++ framework is presented in these proceedings [11].

6 DIM FOR DATA TRANSFERS

When DIM has been operational for the ECS, we realized we had not exploited all its possibilities as a communication medium, in particular in DAS, since we were still using complex interfaces between Fastbus embedded processors and their Supervisors.

6.1 DIM for controlling embedded processors

Thanks to the general API which had been defined for the communications at the OS9 level and at the VMS level, it was very easy to replace the communication libraries using the dedicated hardware interfaces by libraries built on top of DIM. This allows the LES’s and the GES to receive configuration commands from their supervisors through DIM as well as sending back their status and statistics for monitoring purposes.

6.2 DIM for data transfer

Still the LES’s and the GES had to use those complex interfaces to transfer data from Fastbus into VMS where they were handled by a shared buffer system (Model Buffer Manager, MBM).

It was also straightforward to upgrade the data transfer process on OS9 into a DIM publisher. The MBM data producer on VMS is then subscribing to the proper DIM service and gets data over Ethernet into its shared buffer. The total data rate including Central and Local readout for monitoring amounts to 500 kbytes/s which our LAN easily sustains.

6.3 DIM as data distributor

A shared buffer system for data distribution (MBM) was highly valuable when CPU power was low and machines had small memories. This was no more the case in 1998 and DIM was a very good tool for distributing event data.

This has been successfully implemented in 1999 by replacing the MBM Run Time Library by a library based on the use of DIM publishing. Together with a much better reliability of the system, we gained for free the possibility to run data-consuming processes on other machines than the data producer and hence balance the usage of computer resources on our Online Cluster.

6.4 Other possible usage of DIM

Although DIM is currently widely used throughout DELPHI, there are still a few areas where it would have been interesting to use it:

- For communicating between processes and the database server (currently using RPC).
- Error and alarm utility : we use a CERN standard package EMU [12] for error logging. DIM would have simplified logging and error displays at run time.

- Interactive changes of parameters in SC proxies. Currently we still use an old interactive program (HIPE) [13] to modify at run time parameters in the SC proxies. Usage of the DIM command passing would have been profitable in this area as well, as would have been the generalization of DIM publishing of SC settings (currently used by a few sub-detectors only).

7 HOW WAS THIS MADE FEASIBLE?

All these changes were feasible without major modifications to the application software mainly because of three factors:

- Software **architecture**
- Software **design**
- Software **implementation**

7.1 Software Architecture

We spent a long time discussing architecture of the DELPHI Experiment Control System during the years 1985 to 1987. It is at that moment that the concept of SMI emerged and it was decided from the beginning that this framework would be used for an integrated control.

The concepts of supervisors / proxies was well defined with their domains of applications.

7.2 Software design

Before any coding started, the API's of the most important packages in the system have been defined. For the most complex packages such as the Model Buffer Manager (MBM), which was far more general than the use DELPHI wanted to make, we defined a thin layer of software with our own API to avoid programmers to face the complexity of the package. A similar design effort was devoted to define the API for the communications between VMS and the Fastbus embedded processors. A generic API has been defined although at that time no forecast was made yet for replacing the existing hardware interfaces.

7.3 Software implementation

The packages have been implemented using the defined API's as VMS Shared Libraries whose entry points were the routines defined as an API. Straightforward evolution of the software was feasible by simply replacing those entry points.

A typical example is the VMS-Fastbus communications for which three successive implementations were developed: dedicated interface, raw TCP-IP sockets and DIM communications, without changing a single line of code in the application. Same thing for replacing the MBM by a DIM distribution of data.

8 CONCLUSIONS

The DELPHI Experiment Control System is now in 1999 a quite mature system. The usage of our standard packages (DIM and SMI) could have been extended to several domains of the system (Database server, Error reporting…) but due to limited manpower, the existing systems were kept when no added value could be expected (functionality or maintainability). This system was developed over the years without changing the initial architecture nor event the predefined API's, thanks to a careful analysis of the problem leading to an efficient and versatile architecture and design.

9 REFERENCES

[1] DELPHI collaboration, P.Aarnio et al., "The DELPHI detector at LEP" in **NIM A303** (1991) pp.233-276

[2] T.Adye et al., "Architecture and Performance of the DELPHI Data Acquisition and Control System" on **Proceedings of the International Conference on Computing in High Energy Physics '91** (Tsukuba, Japan, march 1991)

[3] T.Adye et al., "The Design and Operation of the Slow Controls for the DELPHI Experiment at LEP", **CERN-DELPHI note 94-14 DAS 151** (1994)

[4] J.A.Fuster at al., "Architecture and Performance of the DELPHI Trigger System" in **Proceedings of the IEEE 1992 Nuclear Science Symposium** (Orlando, Florida, Ocober 1992)

[5] Ph.Charpentier et al., "The Fastbus Inter-segment Processor, FIP" in **Proceedings of the IEEE Nuclear Science Symposium,** 1989

[6] G.Smith, "DELPHI Slow Controls G64 Microcomputers Skeleton Program", **CERN-DELPHI note 94-13 DAS-150** (1994)

[7] J.Barlow et al., "Run Control in MODEL: The State Manager", in **IEEE trans. Nucl. Sci 36** (1989), p 1549-1553.

[8] T.Berners-Lee, in **Proceedings of RT87, IEEE Trans. Nucl. Sci. 34 No 4,** (1987), p 1050.

[9] C.Gaspar and M.Donszelmann, "DIM – A Distributed Information Management system for the DELPHI experiment at CERN", **Proceedings of the RT93 Conference**, Vancouver, Canada.

[10] B.Franek et al., "Big Brother – A fully automated Control System for the DELPHI Experiment", in **Proceedings of CHEP94,** San Francisco,USA, 1994.

[11] C.Gaspar, "An Architecture and a Framework for the design and implementation of large Control Systems", in **these proceedings**.

[12] P.Burkimsher, "EMU, the MODEL Erro Message Utility", **CERN/ECP write-up,** December 1990.

[13] M.Donszelmann, "DELPHI HIPE System User Manual", **CERN-DELPHI note 92-26 DAS-124,** October 1992.