The Evolution of Software Technologies to Support Large Distributed Data Acquisition Systems

Robert John Jones

A thesis submitted in partial fulfillment of the requirements
of the University of Sunderland
for the degree of Doctor of Philosophy

September 1997

School of Computing and Information Systems
University of Sunderland

This research programme was carried out in collaboration with
CERN, European Laboratory for Particle Physics, Geneva
Switzerland

Abstract

Author: Robert John Jones

Title: The Evolution of Software Technologies to Support Large Distributed Data Acquisition Systems

A study of software technologies for the control and configuration of data acquisition systems for high energy physics experiments is presented.

Three key software technologies have been identified that impact the control and configuration tasks, namely, inter-process communication systems, configuration data storage techniques and graphical user interface toolkits.

Investigations and developments of suitable software to meet the unique requirements of large distributed data acquisition systems were carried out at CERN, the European laboratory for particle physics. The research programme was applied to four successive data acquisition system development projects from 1989 to 1997. The thesis describes the evolution of the three software technologies over this period in terms of advances and developments made within the framework of the data acquisition projects.

For inter-process communication systems, the use and relative merits of remote procedure calls, publish/subscribe systems and object-oriented communications complying to the Object Management Group's Corba standard are described and compared. The work on configuration data storage techniques compares a range of technologies from relational databases, to object managers and object databases. In terms of graphical user interface toolkits, various custom-made and commercial software packages are described that provide distributed windowing facilities in a local-area network on a variety of devices including character-cell terminals and bit-mapped workstation screens. A comparison on windowing toolkits and associated graphical user interface builder CASE tools is included.

The work and findings presented are supported by 13 published papers selected from a total of 27 for their emphasis on the three software technologies.

Acknowledgments

The following people have helped the author during his work at CERN. Jean-Jacques Blaising provided access to the material on the use of the MODEL data acquisition system in the L3 experiment and Pierre Vande Vyvre explained many details of the State Manager. I wish to thank Mike Sendall, Sandro Vascotto and Livio Mapelli as the project and group leaders for the data acquisition systems that were the vehicles for this study. Giuseppe Mornacchi has provided continuous technical guidance for many aspects of the work.

I am indebted to Peter Smith and especially Norman Parrington for their support and guidance during the writing of this thesis. I would also like to thank Nick Dyson for proof-reading a draft edition.

Contents

chapter

Introduction

1 Introduction

This chapter defines the scope and purpose of the research programme undertaken and outlines the structure of this thesis. The context and relevance of the published papers is explained and an overview of high energy physics data acquisition systems is provide as background information.

1.1 Scope and Purpose

The main objective of this research programme has been the study of software technologies for inter-process communication systems, configuration data storage techniques and graphical user interface toolkits capable of supporting the ever increasing demands of high energy physics (HEP) data acquisition (DAQ) systems.

These three software technologies have been identified as key components for the control and configuration of large distributed DAQ systems. While other software technologies are important to DAQ systems and are described in published papers referred to but not included in this thesis, the three software technologies mentioned above were emphasised in this research programme since their evolution proved indispensable in satisfying the requirements of modern DAQ systems. Software technologies for the treatment of bulk physics data in hard real-time environments is outside the scope of this research programme.

This thesis is based on existing published work and the research programme is presented via a series of 13 conference papers produced within the framework of four successive DAQ development projects spanning the period from 1989 to 1997. The 13 conference papers were selected from a total of 27 for their emphasis on the three software technologies. Figure 1 summarises the organisation of the published papers in terms of the software technologies they address relative to the DAQ development projects and the timescale of the research programme.

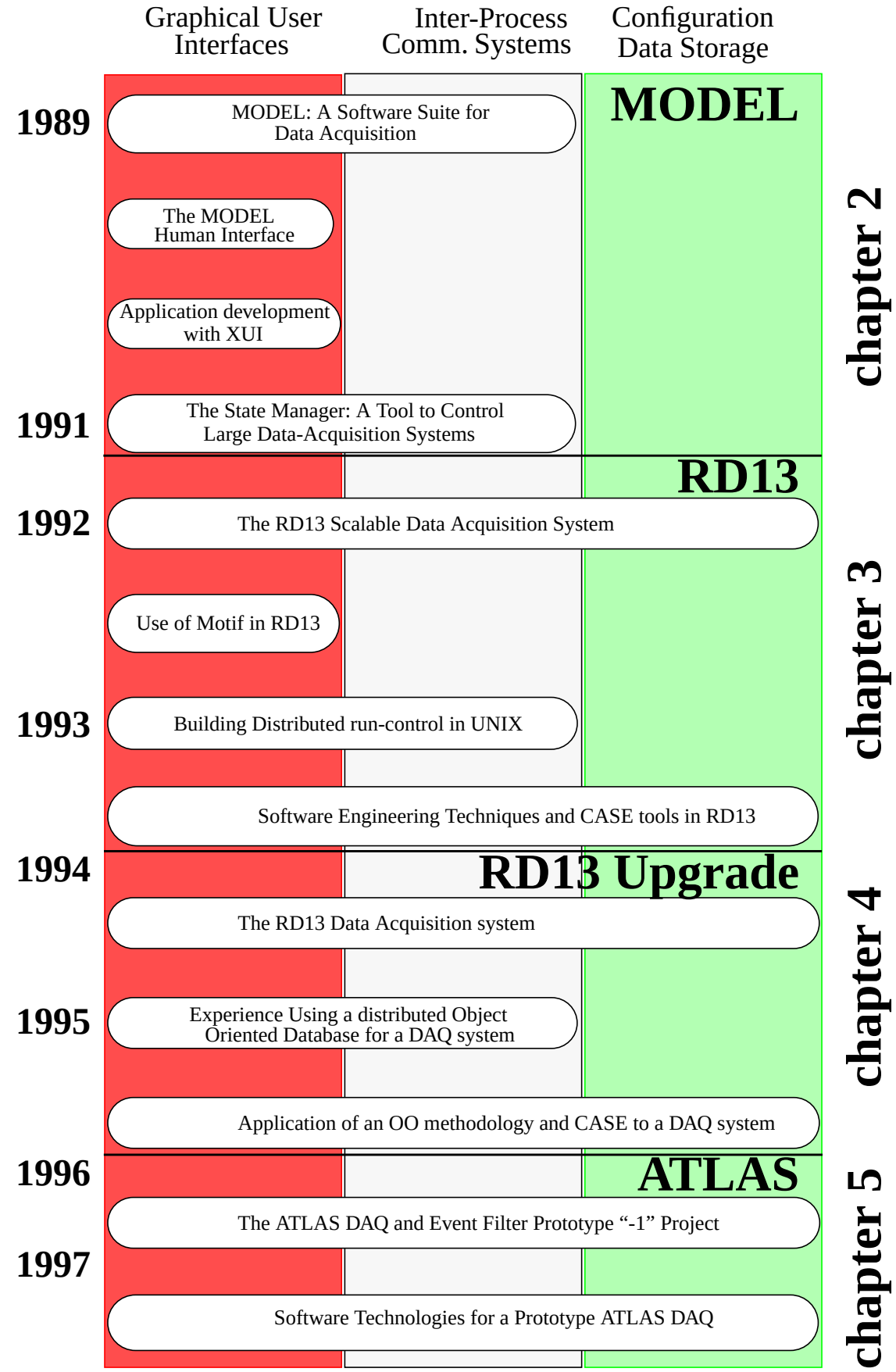
The research programme has been organised into several phases for each DAQ development project:

- understand the essential requirements of HEP DAQ systems in general and the current DAQ project in particular,

- survey existing available packages for each of the three software technologies capable of meeting the above requirements,
- where the survey did not uncover suitable software packages, develop software to meet the specific demands,
- apply the selected or developed software to the DAQ system and exploit it in test-beam or full experimental activities,
- assess the suitability and performance of the software during its exploitation in order to provide guidance for the HEP community at large and the next DAQ development project.

All phases and activities of this research programme have been conducted in collaboration with CERN, the European laboratory for particle physics in Geneva, Switzerland.

Figure 1 Organisation of published papers presented in this thesis



1.2 Structure of This Thesis

The body of this thesis is divided into four chapters corresponding to the DAQ development projects, in chronological order, that provided the framework for investigations into the software technologies, namely MODEL, RD13, RD13 upgrade and ATLAS. The RD13 project has been divided into two projects to reflect the successive developments that were made for each software technology and taking into account the longevity of the project.

For each DAQ project, a published paper provides an overview of the DAQ system and sets the scene for the discussion of the software technologies. The commentary text describes the state of each software technology at the start and at the end of the DAQ project as well as details not covered by the published papers. The software technology published papers of each DAQ project discuss the advances made and outline their application to components of the DAQ system.

The final chapter reviews this research programme and provides a summary of the advances and developments made for each software technology from the start of the MODEL project up to the on going work within ATLAS. It includes a section on contribution to knowledge and makes recommendations for possible future research work.

1.3 Data Acquisition Systems for High Energy Physics Experiments

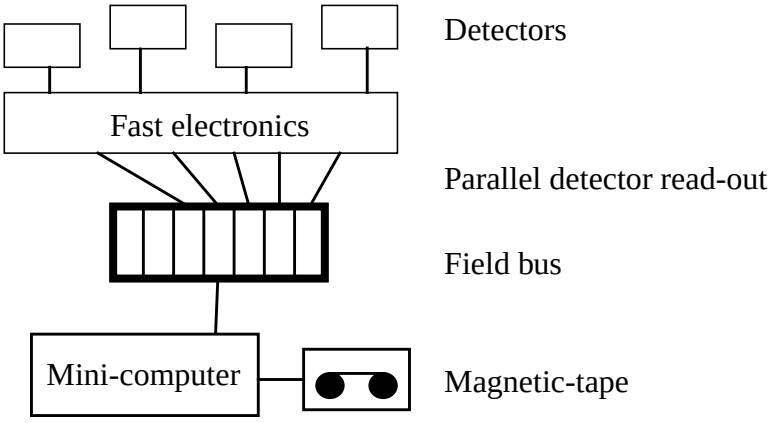
This section provides, as background, an overview of HEP DAQ systems and the role played by their on-line computer systems.

A high-energy physics experiment studies the properties of elementary particles via the interactions of particle beams with stationary targets or with other beams. The acceleration of the particles, and the production and transportation of beams, are specialized fields. Ideally, an experiment would measure the mass, charge, and momentum of all particles but this is impossible for reasons of physics, space, cost etc. and so only the most interesting subset of this information is measured for a given experiment. Computers are used extensively for control functions and data analysis in all major high energy physics experiments.

1.3.1 The Experiment Set-up

A simplified block-diagram (Figure 2) shows the relation between the different parts of the apparatus. The information from the various detectors described below is recorded on permanent storage for later analysis “off-line” on a powerful computer system. The computer system reconstructs particle trajectories, computes momenta and other quantities of interest, distinguishes between different types of events, and produces the distributions of quantities that the experiment was designed to measure. Such analysis can require enormous computing resources, for example it is estimated that to reconstruct a single event for the ATLAS experiment at the future Large Hadron Collider (LHC) accelerator will need 7.5×10^3 SPECint95[1]-seconds.

Figure 2 Typical data acquisition system of the early 1980s



1. SPECint95 is a processor benchmark used as a unit of processing power roughly equivalent to 40MIPS

1.3.1.1 Fast Electronics

The information coming out of the detectors is in the form of analogue pulses that must be treated to give useful information, including:

- Standardization

Discriminators accept pulses from the detectors and for each one above a certain threshold deliver a standard logic pulse of defined amplitude and width. This is an essential first step to digital decision-making and data acquisition.

- Selection

The number of interactions in a beam is very large, and usually only a small fraction are of interest. It is not practical to record everything and analyse it later so selections are made using the fastest detectors. Standardized pulses from these detectors are fed into electronic logic to indicate an event of potential interest has occurred. This procedure is known as the formation of a trigger involving complex selection criteria based on groups of wires in wire-chambers.

- Pulse-Height and Time Measurement

Pulse-height measurement is important since the signal from the detectors can give information on the charge and the velocity of the particle. The signals are sent to analogue-to-digital converters (ADCs) that digitize the pulse-height and store the result in a register. Time measurements give the time-of-flight for a drift-chamber read-out using time-to-digital converters (TDCs) to measure the time between two standard input pulses and store the result in a register.

- Buffering

Buffering smooths-out the random (Poisson) arrival of events and compensates for the duty cycle of the accelerator. For example, CERN’s Super Proton Synchrotron (SPS) takes about 10 seconds to accelerate particles, followed by a short burst (approximately 2 seconds) when they are ejected to provide beams for experiments. The data must all be collected during this short burst, but it cannot usually be processed or recorded in this time and so the time between bursts is needed to deal with it.

- Performance Checking

Performance checking is required to verify the equipment during set-up and to monitor its performance during data-taking. Modern experiments are very complex and accelerator time is expensive and tightly scheduled. In addition the data recorded on permanent storage is often taken away to other labs for analysis and so it is essential that everything is working correctly and that the data does not contain any errors or malfunctions. An example of performance checking is the monitoring of multi-wire chambers where information per channel is gathered and used to construct histograms of wire hits to indicate dead or noisy channels. Another technique is a graphical event display where the apparatus is drawn on a screen and the hits in the various detectors marked. This provides information at a glance about detector performance, beams, background etc.

1.3.1.2 Field Bus

The division between the fast electronics and field bus is often blurred. For example, the trigger logic is usually performed in the fast electronics but the ADCs and TDCs are modules sitting on the field bus. The field bus consists of a number of modules arranged in crates. Each crate has a controller either interfaced directly to the computer or connected together to form a branch and interfaced to the computer via a branch driver. This provides a simple means of exchanging information between the computer and the registers in the modules.

1.3.1.3 Run Set-up and Control

Data taking is organised into runs, varying from a few minutes to several hours long depending on the detectors, trigger conditions, magnet currents, beam energy and many other parameters. The computer provides a convenient means of changing the run conditions. Special hardware exists to provide computer control of the apparatus such as modules to set-up delays in fast electronics or program the trigger logic. Often the experimental computer can communicate with the accelerator control computers via network links allowing it to receive information about beam conditions or to request changes directly. All these general control functions become more important as the experiments become more complex.

1.3.1.4 Sending Data Samples for Remote Analysis

It is unlikely that the experimental computer has sufficient time to analyse all the recorded events completely. Further software is run “off-line” analysing the recorded data more

profoundly. In some experiments, there may be a data-link from the experimental site to the more powerful computer installation used for such off-line analysis. In this case the data can be sent over the data-link to be recorded remotely after greater checking has been performed.

1.3.1.5 Pre-processing the Data

Normally the experimental computer records the raw data on permanent storage and in parallel checks a sample of it. In some applications, however, it may pre-process the data before it is written. Two examples of this activity are data reduction and event filtering. Data reduction involves compressing and re-formatting the data to leave out redundant information resulting in important saving of space and time needed to record an event. In on-line filtering the computer may decide whether or not a particular event is interesting; if not it is thrown away and not recorded. It may be more economical and simpler to do this on-line rather than later. This can be seen as an extension of the trigger logic; a final stage of “software trigger” allows more complex (but slower) checks to be made than is possible with hard-wired logic.

1.3.2 Data Acquisition Systems in the Early 1980s

This section describes the use of computers in HEP DAQ systems in the early 1980s, just before the starting point of this thesis work and during the conception of the MODEL project.

The important characteristics of a mini computer were that it was small, cheap and easily interfaced to special equipment and could be built into an experiment. It could then be regarded as a component of the overall data acquisition and monitoring system. Essentially, the mini-computer added flexibility to the data handling since both the flow of data and its treatment could be easily modified to meet changing circumstances. This made the mini-computer an indispensable part of all high-energy physics experiments of the time.

The typical mini-computers of the period offered instruction times of around 1 micro-second with memory sizes ranging from 16 to 128K 16-bit words. These machines were interfaced to a field bus such as CAMAC [Camac92] (seen as a special I/O device). Peripherals included:

- a keyboard and display for control purposes and to provide visual displays of histograms etc. on demand. It could also be used for software development,

- a printer to record results and error messages,
- a magnetic tape for data recording. Normally experimental data was written on tape and analysed later but in simple test set-ups data recording might not have been needed,
- disks for storing software and large histograms.

A simple test set-up may have contained a small wire-chamber and a few counters, requiring one CAMAC crate with a few modules. Larger experiments could have many chambers with several thousands of wires and hence tens of CAMAC crates containing several hundred modules. The amount of data collected varied with around 1Kbytes being a typical event size.

Fast electronics worked on the timescale of a few nanoseconds. The basic CAMAC operation cycle, and also that of typical computer memories, was about 1 microsecond. This limited the speed data could flow into the computer since it would take a few microseconds to read a full event. The rate at which events occurred depended on the experiment. It could vary from one event every few seconds to far beyond that at which data could be read.

These rates caused some mismatches of performance in the DAQ systems. The fundamental rates of the particle interactions, plus the low dead-time of modern detectors and the speed of fast electronics produced data very quickly. However, it could not be moved into the computer at such rates. Still less could it be processed or even recorded which shows the importance of several levels of selection and reduction as the data flowed through the system. At each stage of reduction more complicated tests were performed but on less data.

The most essential task of the computer was simply to read the data from CAMAC, buffer it suitably and write in to permanent storage. The computer responded to trigger signals indicating an event had occurred and issued the appropriate CAMAC commands to read data into memory. The data was written to permanent storage when a sufficient quantity was accumulated.

Even in this simple activity the flexibility of the computer was useful since it was often necessary to change the details of the read-out (adding new chambers etc.) or the format of the information recorded. The physicists wanted to run the experiment in various ways to

test different parts of the apparatus, calibrate etc., as well as take real data. The computer allowed the physicist to change operating conditions easily.

At the start of the 1980s there were about 100 mini-computers in use by the various experiments at CERN. These minis were mainly Hewlett Packard 2100, PDP-11 or Nord-10 computers. At the computer centre, a CDC 7600 mainframe computer was used for performing off-line analysis. The data acquisition software was normally written in BASIC or FORTRAN.

1.3.3 Data Acquisition Systems in the Late 80s

This section describes the development of DAQ computer systems towards the end of the 1980s when the RD13 project began.

The experiments of the late 1980s were more complex in several ways mainly due to a larger number of channels per detector. Each channel needed to be read out and hence the time taken to transfer data to the computer became significant. More channels implied the cost of electronics per channel needed to be kept low so expensive read-out logic had to be shared between many channels. Checks for low-level malfunctions on the channels were made in parallel by intelligent processors. More channels also implied more raw data was produced per event causing serious problems of bandwidth for on-line data transfer and high off-line processing costs. Thus every effort was made to reduce the number of events recorded and the amount of data per event through complex trigger and filter schemes.

Some modern detectors have intrinsically complex read-out (requiring pulse-shape analysis for instance) meaning the raw data needed to be pre-processed near the chamber before transfer.

The complexity of the experiments raised new problems in the organisation and presentation of secondary data such as calibration data, histograms, status information etc. It was necessary to present to the physicist on shift a meaningful picture of what was happening and allow him to modify the configuration of the experiment and control it accordingly.

Many of the subsidiary tasks in the experiments (e.g. gas and voltage control subsystems, control of fast trigger logic etc.) grew proportionately so they too needed programmable processors.

In addition to these difficulties was the sheer size of the experiments and the number of subsystems. The architecture and components of existing DAQ systems were not able to face these increasing demands. The relatively slow speed at which the mini-computer could execute instructions and so generate CAMAC commands, slowed down the read-out and increased the deadtime of the apparatus. The limited parallelism of the read-out reduced only the part of the deadtime caused by the digitisation process itself. The choice of operating-system was a compromise between the real-time responsiveness required for the event trigger and the subsequent CAMAC read-out against the need to run complicated monitoring programs.

Fortunately, experiments were able to profit from advances in hardware to cope with the extra channels as well as the extra information per channel, such as Flash ADCs, single chip microprocessors with a full 32 bit architecture, Digital Signal Processors, customized chipsets etc. New improved data acquisition buses (such as FASTBUS [Fastbus83] and VMEbus [VMEbus85]) replaced CAMAC and high capacity storage devices became available.

The development of Local Area Networks (LANs) provided a new fast, cheap method of interconnecting processors complementing the bus systems described above. They could be used to interconnect stand-alone computers for which the field bus approach was not appropriate (e.g. due to distance) or to provide an alternative path to microprocessors embedded in the bus systems avoiding contention or bandwidth problems. Personal workstations and specialized graphics devices combining powerful bitmap display subsystems with conventional microprocessors in a single package became commercially available.

The central DAQ computer had several functions, the most important being the recording of the data onto tape. It also provided higher-level monitoring functions since it was the only general-purpose processor with access to the full event data. It had an important role to play in the overall organisation of the experiment: maintaining various databases, making information available to physicists, performing error reporting and status display. It was responsible for the overall initialization and control of the apparatus.

The complexity of the data acquisition systems also placed major requirements on the software needed to operate the experiments. Firstly, the use of distributed systems incorporating hundreds of microprocessors implied more functions and decisions were entrusted to software. In addition the large volumes of data associated with calibration, detector

descriptions and book-keeping required sophisticated tools to manage the on-line database. Facilities were also required to marshal access to all resources belonging to the DAQ. This permitted independent teams to work on their equipment at the same time without interference. The user interface needed to be easy to use and provided as many automatic checking procedures as possible. Interactive graphics stations were used to show event displays and other status information.

To give an example of the size of the problem at hand, some characteristics of the first phase of the ALEPH experiment at LEP were:

- weight: 1500 tons
- volume: 1000m^3
- read-out: 700,000 channels
- event size: 100 Kbytes
- event rate: 10^7 events per year

1.3.4 Summary

The needs of the future LHC experiments represent equivalent increases in detector size and DAQ sophistication as those described between the experiments of early and late 1980s. To control, configure and monitor the operation of such DAQ systems requires sophisticated software. This thesis shows the developments needed in three important areas, namely inter-process communication software, configuration data storage techniques and graphical user interface toolkits necessary to meet the needs of large experiments from the mid 1980s up to the present day.

chapter

The MODEL Data Acquisition System

2 The MODEL Data Acquisition System

2.1 Introduction

This chapter marks the start of the studies into graphical user interface toolkits, inter-process communication systems and configuration data storage techniques. It is set within the context of the MODEL DAQ system and represents the period from September 1989 to April 1991. The author developed two packages while working on the MODEL project: the Occurrence Signalling Package (OSP) for inter-process communication and several components of the human interface package (MHI). The first published paper in this chapter gives an overview of the MODEL software suite and is followed by two papers on graphical user interface toolkits and one on inter-process communication systems. The status of configuration data storage techniques at the time is described in the commentary text.

2.2 MODEL Overview

The MODEL software suite was a set of modules aimed at providing the basic components for data acquisition purposes in HEP experiments. Originally intended for the LEP (Large Electron Positron) collider experiments, MODEL was later used by a wide range of other experiments. Work on MODEL started in 1985 and the software is still in use today in several experiments. MODEL offered a range of software modules that could be integrated into the data acquisition system of a large experiment to complement and extend the services available from the operating system of the experimental computers and the lower-level software. The software suite was designed so that experiments could select only those modules required. Facilities provided by MODEL included data flow organisation, error reporting, man-machine interfaces, software configuration control and run-control. MODEL initially ran under the VMS operating system on computers from the Digital Equipment Corporation VAX family. The typical environment was a set of VAXes, microVAXes or VAXstations linked by an ethernet local area network running the DECNET protocol.

The following published paper, entitled *MODEL: A Software Suite For Data Acquisition* on page 21, marks the starting point of this thesis and sets the scene by describing the software components of a typical data acquisition at the start of the LEP experiments.

Published paper

MODEL: A Software Suite For Data Acquisition

Computing for High-Energy Physics Conference 1989, Oxford, U.K. Computer Physics Communications 57b (1989) 1-7, North-Holland, ISSN 0010-4655

Published paper

2.3 MODEL Human Interface

The MODEL Human Interface (MHI) [Mornacchi87] was designed to provide man-machine interfaces for the control and configuration of on-line systems within the MODEL project. Such on-line systems were distributed in nature and composed of many different display devices thus placing important requirements on the design of MHI.

The MODEL Human Interface (MHI) provided a means of communication between processes running in the on-line system and operators working at terminals and work-stations. It offered a set of packages to help users write interactive, window-oriented applications that were independent of the display device. It supported a distributed environment where an application displays output and receives input from a window on any workstation or terminal in a multi-computer system.

The basis of MHI was a *Window Manager* that could be used either directly by applications or via higher-level *Menu*, *Panel* and *Dialogue* packages. Terminal emulation was supported for Fortran input/output. A GKS (ISO 7942) graphics service allowed applications to use a graphics window on a remote machine. The KUIP [Brun89] interface package was also supported to ease integration with software used for off-line analysis.

The following published paper entitled *The MODEL Human Interface* on page 28 describes the rationale for the MHI package, its architecture and implementation.

Published paper

The MODEL Human Interface

Computing for High-Energy Physics Conference 1989, Oxford, U.K. Computer Physics Communications 57b (1989) 1-7, North-Holland, ISSN 0010-4655

Published paper

Published paper

Published paper

Published paper

2.4 Porting the MODEL Human Interface to the X Window System

The previous published paper described the architecture of the MODEL Human Interface (MHI) and its various implementations. This section describes the issues involved in porting MHI to a new graphics platform, namely the X Window System, via a second published paper entitled *Application Development with XUI* on page 34. This work provided the opportunity to verify as to whether the MHI architecture could be adapted to this new environment by effectively replacing the communication layer between the display device and the application.

In porting MHI to the X Window System, it was also necessary to confront a new programming paradigm. Up until this point, typical DAQ graphical applications were block structured and made discrete calls to windowing packages while retaining control of the execution. The X Window System adopted an event-driven approach where applications submit control of the execution to the windowing package. In order to avoid restructuring all existing MHI applications, it was necessary to find a means of hiding the event-driven nature of X.

Published paper

Application Development with XUI

1989 DECUS Europe Symposium
The Hague, Holland, September 18-22, 1989

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

2.5 Conclusions on MHI

An important feature of MHI was its support for various screen devices. MHI was designed and implemented at a time when bit-mapped workstation screens were just being introduced and no clear windowing graphics standard existed. MHI helped to bridge the gap between radically different device types and allowed graphical applications their first possibility to exploit processing power in a distributed environment.

The effort required in porting MHI to new platforms could have been reduced by restricting the functionality of the Window Manager layer that interfaces with the device specific software. This would have meant moving more intelligence into the various packages layered on top of the Window Manager. In order to support character-cell terminals in the UNIX environment, it would have been possible to port the Window Manager to the Curses [Goodheart91] package which is similar to SMG.

Despite many advances in this field, MHI continues to be used for a subset of applications in various experiments [Balestra91], primarily because of its support for character-cell and bit-mapped screens. Character-cell terminals are often preferred in certain experimental areas subject to strong magnetic fields since workstations require special shielding to protect screen images.

In order to provide a template for graphical applications in the L3 experiment, another layer of software was developed on top of MHI by the experiment's staff. This encompassed the facilities provided by the Window Manager and Panel Package and was later rewritten [Wenaus89] to use the SMG and UIS packages directly with the intention of simplifying the software.

Perhaps the strongest attribute of MHI was its basic architecture that combined a client-server model with device independence. This allowed MHI to be ported to the X Window System.

There are many parallels between the development of MHI and the X Window System that has become the standard for UNIX workstations. Both offer a client-server model with RPC-based communication systems for providing networking capabilities and during the porting of MHI to X Window it was possible to drop CERN's RPC package for communication and use the X Window System's own communication protocol.

However, a fundamental difference between the X Window System and MHI is that X is event based and so application code is only executed in response to signalled events. In comparison, MHI was a passive library of routines where the application retained control. This difference was partially related to the lack of asynchronous I/O facilities in the UNIX environment where X was developed. A mainloop is required in UNIX to receive and dispatch events whereas Asynchronous System Traps (ASTs) could be used under VMS allowing the basic control flow to remain with the application.

The development of the *Exposure* process in MHI's X Window System port showed a limitation the X Window System itself. The fact that applications are responsible for repainting damaged screen contents is a means of easing the implementation of the X server and insulating it from an exhaustible resource, namely storage capacity to hold copies of window contents or display lists to replay and rebuild the contents.

The facilities provided by MHI also included those of X Window System window managers. Such X Window System window managers are just like other clients except that they usually re-parent windows created by others in order to decorate them with borders and manipulation facilities. The X Window System's manner of separating window manager functions from basic windowing facilities simplifies the implementation of the server but the act of re-parenting windows causes confusion when applications inquire about the window hierarchy. Such facilities shows a limitation of MHI - it was ignorant of non-MHI applications that used the same display device.

The facilities provided by the layered packages (i.e. menus, panels and dialogues) are now supported by today's most common toolkits namely Open Software Foundation's Motif and Microsoft Foundation Classes. The advent of graphical interface builder CASE tools for the most common toolkits (e.g. X-Designer for Motif and Visual Basic for MFC) has simplified the task of graphical interface development. But a commercial market for such GUI builders can exist only when a common standard for graphical toolkits has gained wide acceptance. This was not the case when MHI was designed and implemented. It would have been possible to develop an interactive screen painter or GUI builder for MHI, for example implemented in MHI itself and generating Fortran code with embedded calls to the various MHI packages.

As described in the following chapters, support for graphical user interfaces has continued to improve in HEP DAQ systems. The following chapters include details of the early application of advanced GUI packages to DAQ systems that have shown the direction for upgrades of applications of MHI and other packages. The work on GUIs progressed further when the opportunity arose for the author to develop a graphical interface to the run-control system as described later in this chapter (Graphical User Interface for the MODEL State Manager on page 57).

Recently, the Java programming language [Arnold96] together with the Abstract Window Toolkit (AWT), has become widely available on many platforms. Although Java is not restricted to graphics applications, it offers many of the features of Motif and MFC while retaining the networking capabilities of the X Window System. Technically, it offers the possibility of making truly platform independent interfaces but more time is required to determine if it will prove to be a reliable, widely-supported, long-term alternative to the combination of the X Window System and Motif toolkit.

2.6 Inter-Process Communication Systems

Every module of the software suite needed a means of communication between its internal elements, with other MODEL modules and experiment specific software. The most popular means of communication within the MODEL suite was the Remote Procedure Call [Berners-Lee87] (RPC) for point-to-point, client-server, synchronous connections. However, for certain modules and applications it was found that RPCs were not the most suitable form of communication. One particular example was the run-control system as implemented by the State Manager.

2.6.1 The State Manager

The State Manager (SM) was the MODEL module for run-control. Run-control implies the execution and synchronization of the various procedures needed to begin, maintain and end a period of data taking under stable conditions. The requirements vary not only from experiment to experiment, but may need to be substantially modified within a particular experiment as setting-up proceeds and experience is gained. Other aspects, such as calibration or cold start of the apparatus, pose similar problems. In general any change in working conditions requires interventions in various parts of the data acquisition system. These normally consist of lists of actions to be executed by the operators running the experiment. An overview of SM is given in the following published paper entitled *The State Manager: A Tool to Control Large Data-Acquisition Systems* on page 48.

Initially SM used the RPC package for communication with associated processes. The use of RPCs had to be abandoned because the RPC style of communication, as was available at the time of implementation, did not satisfy the requirements for SM control messages. These requirements are now discussed.

The SM's associated processes (i.e. processes that control individual devices) might connect and disconnect from the SM server at any time. Frequent reconnections by new incarnations of associated processes posed problems for the RPC server. Also, RPC server addresses had to be known at compile-time whereas SM was more dynamically configurable implying that such information was not available until run-time. An SM server process supported many SM objects implemented as individual Ada tasks. For each object, a separate message queue was required to marshal incoming commands but

an RPC server needed to be a singleton and could not support multiple SM objects and message queues.

In more general terms, the use of RPC was not quite as transparent when compared to local procedure calls as one might hope. RPC systems usually put restrictions on the procedures that can be invoked remotely by limiting the number, type and size of the parameter list. For example, the use of pointers as parameters is severely limited since they have no mapping into the address space of the remote procedure.

If in a complex RPC system based on single-threaded programs, a cycle formed where client A calls server B which needed some service C that is also implemented by client A then a deadlock occurred. In a local procedure call implementation, such deadlocks need not occur since recursion could have been used.

2.6.2 The Occurrence Signalling Package

In order to implement the communication scheme required by the State Manager that addressed the limitations of the RPC system, the author developed the Occurrence Signalling Package (OSP) [Jones86] as referred to in the published paper entitled *The State Manager: A Tool to Control Large Data-Acquisition Systems* on page 48 and described below.

2.6.2.1 Overview

OSP offered a service to meet the needs identified for communication within the MODEL software suite [Sendall86]. OSP allowed client programs to be notified of user defined occurrences as they happened. An occurrence was a software event generated by other clients. OSP allowed a client to signal an occurrence that was then distributed by the OSP server to all the clients who had expressed an interest in the occurrence (including the signalling client if this was the case). The occurrence name was a user defined string passed to the interested clients. All clients had a unique identification string. The string for identifying the client who produced the signal and a variable length buffer of optional information (defined by the source client) were also sent with the occurrence name.

Clients expressed an interest in an occurrence by either calling a function that allowed them to wait for an occurrence or specify an action routine to be called when the occur-

rence was signalled. A wait call received notification of an occurrence once only but a call that associated an action routine with the occurrence received a notification every time the occurrence was signalled.

To allow clients to wait on or associate an action routine with several occurrences at the same time, the concept of an occurrence group was introduced. The client declared an occurrence group into which occurrences were inserted. Group-associate and group-wait functions operated similarly to the individual occurrence-associate and wait functions and returned the same information. Occurrence groups had only local significance within the client program. Functions existed to allow the client to insert and delete occurrences to and from groups.

The OSP Server

The OSP server was implemented as a continuously running network daemon process that waited to receive network messages. It held information about clients and the occurrences in which they had expressed interest. It informed the appropriate clients when a matching occurrence was signalled and also handled the initialisation and termination of an OSP session for clients as well as exception conditions (e.g. collapse of a client, network error codes, etc.)

The OSP Client

Clients linked their application programs to a library that contained all the OSP function calls, handles all the communication with the server and maintained a local client database of information about occurrences and occurrence groups. The client was continuously listening for messages on its dedicated network channel and mailbox. OSP was implemented in VAX-11 PASCAL V2.

The following published paper entitled *The State Manager: A Tool to Control Large Data-Acquisition Systems* on page 48 describes the MODEL State Manager and refers to the OSP package.

Published paper

The State Manager: A Tool to Control Large Data-Acquisition Systems

International conference on accelerator and large experimental physics control systems, Tsukuba, Japan, KEK Proceedings 92-15 (524-527).

Published paper

Published paper

Published paper

Published paper

2.6.3 Conclusions on OSP

Apart from the State Manager, OSP has been found useful in many other applications. OSP applications have been developed by users to monitor and control on-line systems in several experiments including DELPHI [Adam91], CPLEAR [Bee92] and L3 [Angelov91]. To simplify the interface between SM and OSP, an intermediate package called ICT [Vascotto89] was developed. The architecture of the OSP package offered several advantages over simple point-to-point RPC style connections:

- Each client needed only a single network connection to link it to the server. This economized on machine resources when there were many clients wanting to communicate with each other. It was not unusual for a LEP experiment's on-line system to be composed of 200 or 300 processes running on 10 or more computers,
- A single server model implied synchronization issues were simplified. In contrast, if OSP used multiple servers they would have needed some form of internal communication protocol and synchronization to ensure that the client databases in each server were consistent and that occurrences were reported to all clients in the same order,
- The ability to partition the service either by using multiple, unrelated servers or simply using naming conventions for clients and occurrences (as was the case in the ICT package) allowed OSP to be used to define disjoint name spaces,
- Group based communication offered the possibility to dynamically change the set of receivers for an occurrence without any modification to the sender. This allowed easier reconfiguration of the application and the possibility of adding extra listeners for debugging or monitoring purposes,
- The simple client-interface provided by OSP allowed it to be used in a diversity of applications and ported to new environments such as UNIX and real-time kernels including OS-9 [Microware97] [Microware96]. To ease portability issues, the OSP implementation was translated from VAX-11 PASCAL into C. DECNET was replaced with TCP/IP. This work showed some limitations of traditional UNIX implementations. UNIX has no provision for asynchronous input/output operations as provided by the ASTs under VMS. The lack of asynchronous I/O meant that the UNIX implementation had to implement an event loop for handling and distributing signals indicating incoming network messages. This fundamentally changed the structure of an OSP client. TCP/IP, unlike DECNET, does not perform message

queuing so OSP had to provide a system of message queues that would be treated whenever the package was given control of the process. OSP became less reliable as a result of these changes. When a message queue became full OSP could either block the application until the queue diminished or throw away excess messages.

OSP Limitations

It was stated earlier that the decision to use a single server model simplified the implementation of OSP and avoided many synchronization issues. But this model also introduces certain limitations:

- A single server was a single point of failure. If the OSP server failed then all applications were immediately affected since no communication was possible between any clients,
- There was no provision for restoring information about existing connections when the server restarted,
- A single server limited the scalability of the system. Each OSP client had only one network connection but the server needed one connection for each concurrent client. The default limit on most operating systems is 64 or 128 connections per process but this could normally be increased via a kernel parameter. As more clients connected, the server needed more time to search its client database whenever an occurrence was signalled hence degrading performance,
- DECNET (and traditional TCP/IP) does not offer true broadcast communication. The OSP server simulated broadcasts of occurrences by serially sending asynchronous messages to the list of receivers. Hence the time required by the server to signal an occurrence increased in proportion with the number of receiving clients,
- All OSP messages were ASCII strings. The use of ASCII strings avoided conversion problems when applications spanned computers of different architectures and operating systems. However, many applications also needed to send non-ASCII data which, in the OSP model, required the data to be converted to ASCII by the sender (thereby significantly increasing the space requirements) and unpacked by the receivers. This was a serious limitation on the potential performance of OSP for sending bulk numeric data. The conversion was the responsibility of the user's code so the applications needed to adopt a convention for packing/unpacking messages.

Communication Styles

Within HEP software, the OSP package introduced a new style of communication in distributed applications. The majority of HEP software used point-to-point RPC type communication but OSP opened the possibility to develop another range of applications that extended the functionality of data acquisition systems. OSP permitted three distinct kinds of interactions to occur among applications:

- Request/Reply interactions, such as queries as transactions,

This is the only type of communication offered by traditional RPC systems. In request/reply interactions data producers coordinate closely with data consumers. A producer does not send data until a consumer makes a first request. Producers send replies specifically to the client that requested the data. The requesting client listens until it receives the reply, and then stops listening.

- Broadcast request/reply interactions, such as queries that may result in several replies from one or more servers,

In broadcast request/reply interactions, as in point-to-point request/reply, producers do not send data until a consumer makes a request. If a producer has the information the consumer requested, the producer sends a reply specifically to the client. The requesting client listens until it receivers one or more replies. The client stops listening when it has received sufficient information.

- Publish/subscribe interactions, such as general distribution of information from many sources to many consumers,

Publish/subscribe interactions are event-driven (rather than demand driven). In this paradigm data producing applications disseminate data to multiple data consuming applications. Publish/subscribe interactions are driven by events (usually the arrival or creation of data) in the producer component. Communication is in one direction only, and is often one-to-many. In publish/subscribe interactions data producers are decoupled from data consumers since they do not coordinate data transmission with each other. Producers publish data to the network at large. Consumers can receive messages with any subject names. Any application can be both a producer and a consumer. This event-driven paradigm is a more natural, more responsive, and more efficient style of computing for many applications and is likely to emerge as the dominant paradigm.

2.6.4 Related and Further Work

In order to implement the OSP server it was necessary to develop a multi-client server in DECNET. Such technology had not been used in the implementation of CERN's RPC package. On reflection it became apparent that if such technology were available in the RPC system then it could be considered as the means of communication in MHI to allow multiple programs to talk to the same screen server and in the process manager (MPC) to make a shared database server. The author implemented the RPC multi-client server on DECNET that became part of the standard product and allowed RPC to be used in a wider context.

OSP proved extremely popular as a inter-process communication package in many experiments both at CERN and other laboratories. As a consequence of its popularity, experiments have developed new inter-process communication packages that take advantage of their specific installations:

- The Cluster Process Communication package (CPC) [WenausT89] was designed for the L3 on-line data acquisition system. The L3 on-line computer system consisted of a single large mixed VAXcluster of about 30 nodes so CPC used the proprietary Distributed Lock Manager [Dec87] from Digital as a basis for communication,
- DELPHI's Information Manager (DIM) [Gaspar93] was developed so that user interfaces, control and monitoring processes could have access to all the information produced in the on-line system.

2.6.5 Inter-Process Communication in the RD13 Project

The need for an inter-process communication facility similar to OSP was identified in the RD13 project. Based on the advantages and limitations discussed above, a survey of alternative technologies was made. As a result of this survey, it was decided to investigate the ISIS fault-tolerant distributed communication package from Cornell University as a replacement for OSP and some of its layered applications. This work is described in the published paper entitled *Building Distributed Run-Control in UNIX* on page 88 in the next chapter.

2.7 Graphical User Interface for the MODEL State Manager

As a means of linking together the studies of inter-process communication systems for control purposes and graphical user interface toolkits, the author developed a Motif based user interface to SM. The need for a general purpose graphical user interface during the development, analysis and the control of SM applications was identified by the experiments using the product for run-control purposes. Such an interface, called the SM display program, had been developed at the DELPHI experiment based on MHI. This program was written in Fortran and communicated with the SM server via a common block. This implied the display program had to run on the same machine as the SM server and it was closely tied to the SM configuration in use at DELPHI. A display program independent of the configuration yet capable of displaying the status of SM objects, modifying them and providing debugging facilities was needed.

Having ported MHI to the X Window System, it became evident that the basic X toolkit did not provide the high-level graphical objects required for such an interface. At this point in time (1990), initial versions of higher-level windowing toolkits were becoming available from workstation vendors, such as Digital Equipment Corporation [Dec88], of what later evolved into the Open Software Foundation's Motif toolkit [OSF93]. Digital also made available an early version of a graphical interface builder CASE tool, called VUIT [Dec91], that could be used to interactively paint screens and generate the corresponding structure to an ASCII file according to the User Interface Language (UIL) syntax. At run-time, the UIL definition could be combined with C code to make a complete interface for the application.

The author used Digital's toolkit and a beta release of the VUIT GUI builder to produce the SM display program mentioned above and referred to in the published paper entitled *The State Manager: A Tool to Control Large Data-Acquisition Systems* on page 48. This work demonstrated convincingly the advantages, in terms of abstraction and reduced source code size, of using a high-level windowing toolkit. The use of VUIT, though only a beta release, allowed for rapid prototyping and an overall reduction in the development time.

However, these tools were proprietary to Digital Equipment Corporation and the application would require modification at the source code level to operate on alternative plat-

forms. This work, together with the development of MHI, lead to the selection of the X / Motif configuration and the X-Designer GUI builder for the interface work described in the following chapter on the RD13 project.

2.8 Configuration Data Storage

This section gives an overview of the uses of configuration data in an experiment and the DAQ in particular. It then goes on to describe the state of configuration data storage within the MODEL project.

2.8.1 Configuration Data Storage Needs in a HEP Experiment

Within a HEP experiment, typical types of configuration data to be stored include:

- geometrical detector description

models of the geometrical structure of the detector are needed during construction, integration and by off-line reconstruction software.

- read-out electronics description

describes the structure, composition and internal functionality of the read-out electronics and field bus for the detector. Such a database has been developed for FASTBUS based read-out systems [Rimmer87]. An example query could be *"An error has been detected in read-out crate 21. Which part of the detector is affected?"*

- electronics and material book-keeping

tracks all the electronics modules and materials used in an experiment. It contains information such as module description, manufacturer, purchase details, inventory and location. It is useful to have a link to the read-out electronics database. An example query could be *"Do we have any spare VME 8351 processors from the CES company available on-site?"*

- experiment book-keeping

In a large experiment it is difficult to trace a single physics event through the various phases of the off-line analysis. This data store keeps track of the physics data files, program versions with which the data files are compatible as well as the media used for their storage (e.g. disks and tapes). It must be accessible by the reconstruction software in all the institutes of the collaboration. An example query could be *"Give me all the Z0 events found between June 21 and August 2"*.

- calibration constants

calibration constants are corrections to experimental data and represent a huge amount of unstructured data. A link to the geometrical description of the detector is required. An example query could be *"What are the calibration constants for run number 4631?"*

- experiment directory

contains all the administrative information necessary to permit communication between all the members of the collaboration such as address, phone number, office, function, responsibility, affiliation etc. An example query could be *"Which members of the collaboration from the St. Petersburg Institute are participating in the development of the calorimeter?"*

2.8.2 Existing Data Stores Used in HEP

In the past, a number of techniques have been used in HEP experiments to handle the large amount of data needed for the data acquisition (on-line) and reconstruction (off-line) programs. These ranged from hard-coded DATA statements in Fortran programs to the sophisticated usage of external files containing the necessary information accessed by programs at execution time. But not even within one experiment could people agree on a single access method. This scenario is no longer valid, given the size and time-scale of present-day experiments.

As late as 1985, the subgroup of the ECFA Working Group on Data Processing Standards [Blobel83] stated that commercial database management systems (DBMS) were not used in HEP experiments. The most commonly cited reasons for not adopting commercial DBMS for scientific databases included specific additional requirements such as the following:

- Scientific data processing tends to be based on large data sets or entire tables at a time rather than transactions on individual records,
- Locking and security issues are normally of less importance than in commercial data processing and scientific databases are frequently read-only without multi-user updates,
- Scientific data types are normally not well supported (e.g. true floating point numbers with adequate precision and vector and matrix formats),

- Procedural languages are preferred to SQL since it lacks support for scientific and user-defined data types and user functions.

But the ad hoc systems designed to meet specific needs were not flexible enough to cover all the requirements of an experiment and so this situation has since changed. However, most experiments have found it necessary to develop their own software on top of the basic DBMS to make it more friendly and overcome some of their database deficiencies. A summary of the packages used by the LEP experiments is given in [Rimmer90].

2.8.3 Configuration Data Storage Within MODEL

A number of components within the MODEL DAQ system needed some means of storing configuration information. Each component adopted its own techniques for storing and accessing such information but these were generally incompatible. A study was performed to understand how the introduction of a consistent data storage technique across all the components could be used and suggested the following possibilities:

State Manager (SM)

Inside the experiments, a data store was already being used to configure the FASTBUS read-out system [Rimmer87] and the users would have liked to extend this to include all the hardware devices and all their software attributes, relationships and states. Analysis of the application showed that it might have been possible to control SM object states using a list of permitted states for the objects to draw on. An advantage of controlling states is the resulting consistency, for example, *run* and *error* would have the same meaning for a tape device and application program. Once held within the data store, the definitions of all such objects could have been used as input to SM itself. A program could scan the data store and generate an input file for the SM translator. As a second step, SM could retrieve the information itself from the data store. SM accepted parameters as VMS operating system literal or logical names to be translated. These parameters could then be passed to SM objects. This scheme could be extended so that such parameters could also be the names of fields in the data store.

Process Manager (MPC)

MPC maintained its own data stores with information processes, machines, users and terminals to provide its services but this information could not be shared with other modules.

Buffer Manager (MBM)

The configuration of the ports, channels and buffer sizes could have been defined in the data store.

2.9 Conclusions on the MODEL Project

The MODEL software suite made substantial contributions to the on-line systems of three of the four LEP experiments. It has been widely used in other experiments both at CERN and other laboratories.

The MODEL project marked the starting point for the author's investigations in the area of graphical user interface toolkits and inter-process communication systems and provided the occasion to survey the current usage of data stores in DAQ systems.

2.9.1 Graphical User Interfaces

For graphical user interfaces, The author learnt that it was important to provide a network-enabled windowing facility since DAQ systems were becoming more distributed. At the same time, the display devices and processors on which graphical applications were to run diversified and hence it was important to find a vendor and device neutral standard for basic windowing facilities. The X Window System offered such a standard but was too low-level to provide all the facilities required by typical DAQ applications. This work continued with the investigation of the Motif toolkit in the context of the RD13 project.

2.9.2 Inter-Process Communication Systems

The limitations of the simple Remote Procedure Call approach became apparent via its aborted use in the run-control system. As an alternative, the author investigated the publish/subscribe approach via the development of the Occurrence Signalling Package. This work showed the basic principles were sound but that the implementation was too closely tied to the operating system, networking services and programming language. This work continued with the investigation of the ISIS system to develop the run-control and error message facilities in the context of the RD13 project.

2.9.3 Configuration Data Storage

No consistent approach had been adopted for configuration data storage in the MODEL project. Each component had implemented a private technique for its own needs. An investigation was made as to how the project could benefit from a configuration data storage service.

chapter

The RD13 Data Acquisition System

3 The RD13 DAQ System

3.1 Introduction

This chapter describes the studies performed on the software technologies within the context of the second DAQ system, called RD13 DAQ, during the period April 1991 to December 1993.

The first published paper provides an overview of the RD13 DAQ project, followed by papers on graphical user interface toolkits and inter-process communication systems. A final paper explains the use of configuration data storage techniques and describes the state of the research in all three software technologies at the end of the first phase of the project.

3.2 RD13 DAQ Overview

The RD13 project was approved in 1991 to develop a scalable data taking system suitable to host various studies for LHC experiments [Mapelli90]. At the outset of the RD13 project, it was possible to build on a number of principles established by the precedent MODEL project.

For graphical user interface toolkits (GUIs), the layered structure of the MODEL Human Interface (MHI) had proved successful and was confirmed by the large scale use of the X Window System which has a similar architecture. The upper layers of MHI, namely the forms, menu and dialogue packages, were the most appropriate for the majority of applications and their equivalents were available as toolkits built on top of X although there was still no clear standard. Access to early versions of the VUIT GUI builder had shown that such CASE tools were useful for rapid prototyping and could significantly reduce GUI development time.

For inter-process communication systems, the Remote Procedure Call (RPC) had been used extensively but had proven inadequate for a class of data acquisition applications, such as run-control. As a consequence, the author developed the Occurrence Signalling Package which proved popular but its implementation showed limitations in terms of portability, scalability and fault tolerance.

No clear policy had been established for configuration data storage in the MODEL project. Various approaches had been explored but the majority consisted of application specific data files. However, the need for a consistent data model representing information of interest to several components of the DAQ had been recognised and limited tests with the ORACLE relational database management system had been made.

The following published paper entitled *The RD13 Scalable Data Acquisition System* on page 67 describes the rationale for the RD13 project and gives an overview of the DAQ architecture.

Published paper

The RD13 Scalable Data Acquisition System

Computing for High-Energy Physics Conference 1992, Annecy, France ISBN 92-9083-049-2.

Published paper

Published paper

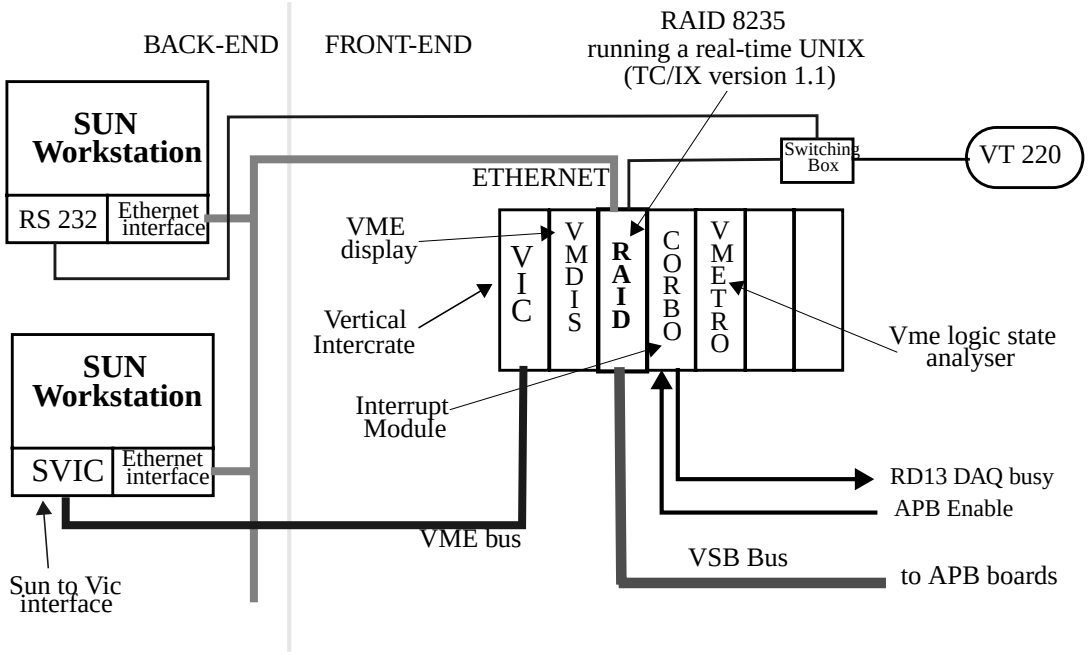
Published paper

Published paper

3.3 Test-beam Activity

The first version of the RD13 DAQ was in a test-beam set-up for the SITP (RD-2) detector R&D in November 1992 during 10 days when approximately 5 million events were recorded [Mapelli92]. This system was based on VMEbus, using the VICbus (Vertical Inter-Crate) to link VME crates and to integrate back-end workstations (sun SPARCstations and HPs) with the front-end processors (MIPS 3000 based CES RAID 8235 boards) via the SVIC (Sbus to VIC interface). All the processors (front-end and back-end) ran UNIX as the common operating system. A real-time version of UNIX (EP/LX, a port of LynxOS to the MIPS architecture) proved suitable for use in the front-end RISC processors. This implementation [Mapelli93] was used with the TRD (RD6) detector during 1993 test-beam periods (Figure 3).

Figure 3 The RD13 DAQ hardware configuration 1993



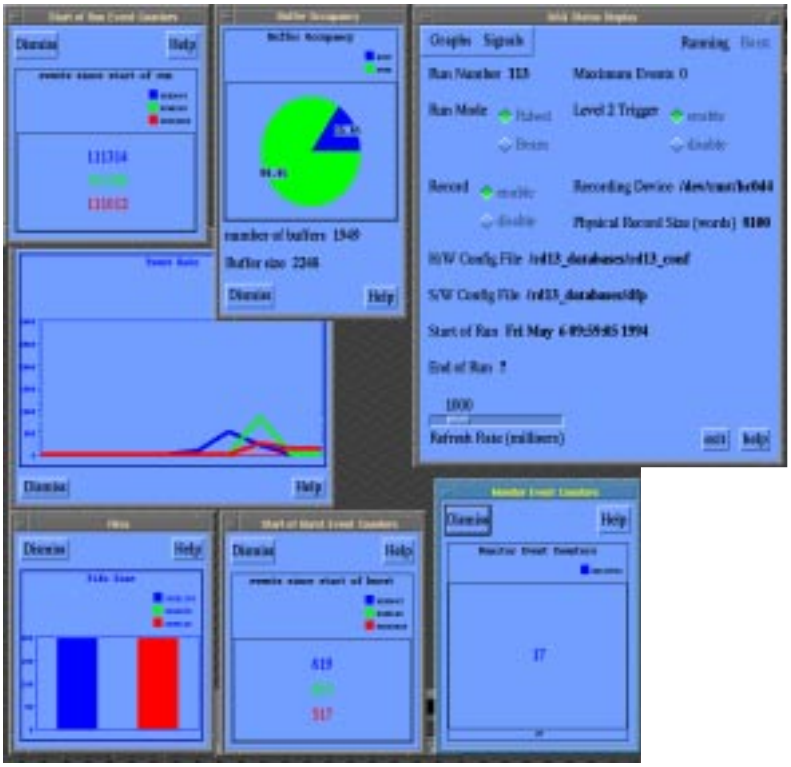
3.4 Graphical User Interface Toolkits

At the outset of the RD13 DAQ project, the X Window System had established itself as the most popular basic windowing facility but two higher-level GUI toolkits competed heavily in the marketplace: Sun Microsystem's OpenLook and OSF's Motif. The following published paper entitled *Using Motif in RD13* on page 76 describes the selection criteria employed for choosing the X Window System and the Motif toolkit instead of OpenLook. The practical difference between these two toolkits was that Motif gained popularity because it was available on virtually every UNIX client platform (including

Sun workstations), while OpenLook remained a proprietary product from Sun. Motif offers a set of general purpose graphical objects and specialised ones for performing tasks like file selection. Such widgets were useful in the development of user interfaces to components such as the run-control (Figure 8).

During the work a number of areas of interface design and implementation were found to be not adequately addressed by the Motif toolkit. In all cases, it was possible for the developer to work around the problems by either implementing the required functionality using the toolkit or by acquiring it from other sources, be it commercial or shareware. The published paper entitled *Using Motif in RD13* on page 76 explains these limitations. For example, the graphical event dump (Figure 6) called for the use of specialised tree and table widgets whereas the status display (Figure 4) required frequent updates of business style graphics.

Figure 4 DAQ status display built using X-Designer with Motif and DataViews widgets



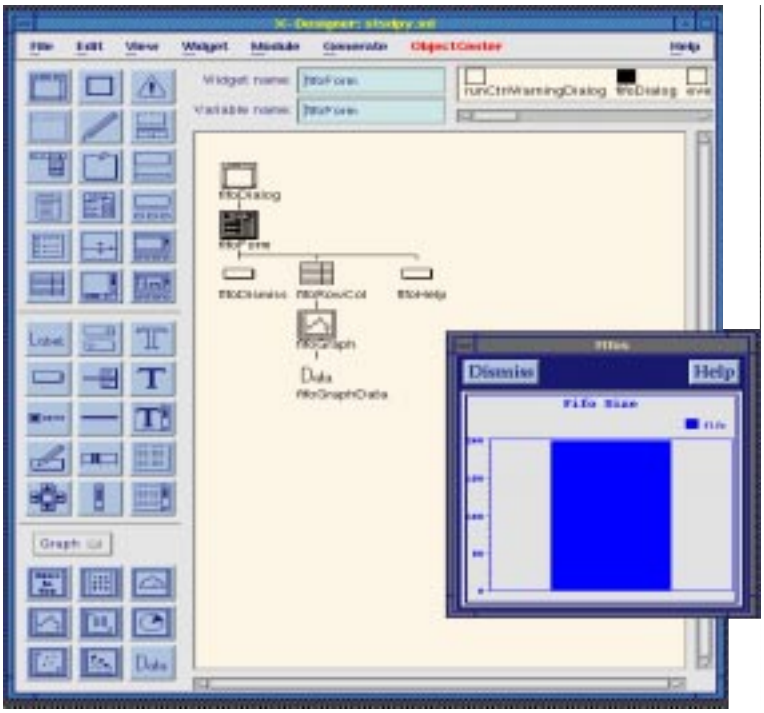
3.4.1 Graphical User Interface Builder

Graphical user interface builders allow the developer to interactively design interfaces then generate code for their implementation. Using an interface builder the developer can implement an interface far more quickly than by hand coding calls to the toolkit.

Interfaces can be built in a more abstract manner and the developer need not have a deep understanding of the toolkit itself.

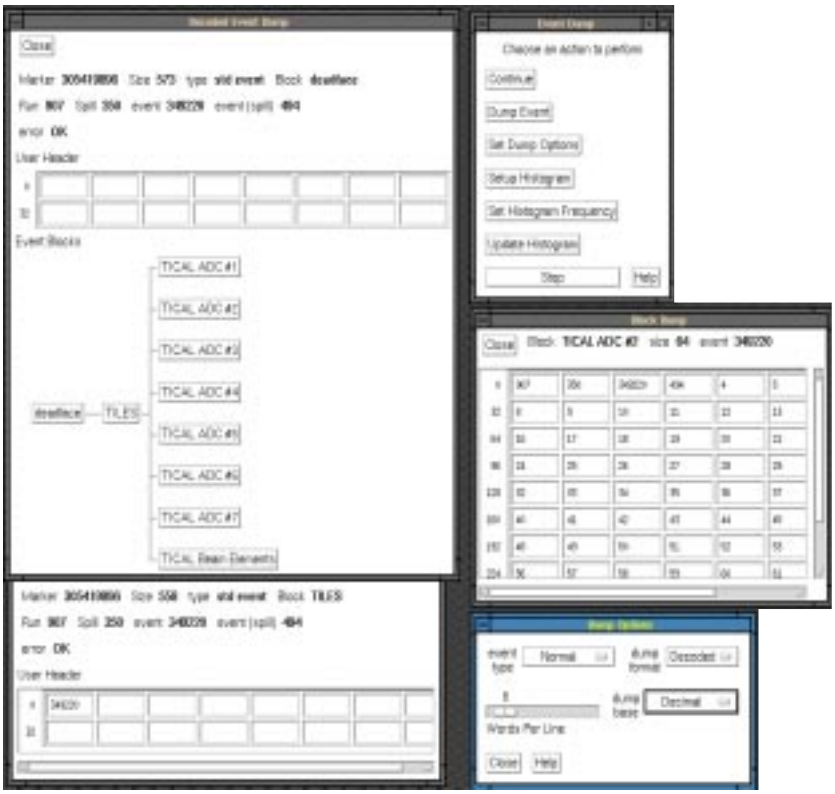
For the development of the DAQ, a commercial graphical user interface builder called X-Designer (Figure 5) (Imperial Software Technology, U.K.) was chosen and proved invaluable in aiding the production of user interfaces. X-Designer offers a means of incorporating 3rd party widgets (including the DataViews widgets) and allowing them to be used along side the standard Motif widget set.

Figure 5 X-Designer graphical user interface builder with Motif and DataViews widgets



An example of a monitoring task with a graphical user interface was the event dump (Figure 6) built using X-Designer. The event dump displays the contents of an event and is a useful debugging tool allowing the physicist to verify the format of the data from the detector without waiting for a tape to be written and analysed.

Figure 6 Event dump showing event decomposition and data block contents



The field of graphical user interfaces is evolving rapidly and has moved forward since the paper entitled *Using Motif in RD13* on page 76 was published. The paper contains references to GUI builders and widgets available at the time. Eventhough the details of the information are out-of-date, the selection criteria established in the paper are still valid. Of the toolkits mentioned, Motif has developed into the most popular choice for the UNIX platform while Sun's support for OpenLook has declined and InterViews has been superseded by another project called Fresco [Fresco94] that uses a standard object model (OMG CORBA) for transparent distribution of user interface components.

As a further development of this work, it became clear that the design and implementation of the relationship between the GUI and the application itself needed better support. This issue is addressed by the incorporation of a GUI builder in an integrated development environment called the Object Management Workbench described in the next chapter. The issue was further pursued by the adoption of the Model-Viewer-Controller design technique for GUI interaction in the subsequent chapter on the ATLAS experiment.

Published paper

Using Motif in RD13

Second International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics (AIHEP'92) conference, La Londe-les-Maures, France in January 1992. ISBN 981-02-1122-8.

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

3.5 Inter-Process Communication

During the preceding MODEL project, the author developed the OSP inter-process communication package that was used by the SM run-control component and many other applications. In the RD13 project, the author's work on the run-control component provided the means for continuing investigations into inter-process communication systems. Based on the limitations of the OSP package described earlier (see *Conclusions on OSP* on page 53), a survey was made of available products which resulted in the selection of the ISIS toolkit.

ISIS [Birman90] [Birman96] is a commercial toolkit for distributed and fault-tolerant programming. The toolkit is a set of fault-tolerant software protocols that are accessed by an application programmer interface. Support is included for groups of cooperating processes, replicated data, distributed computation and fault tolerance. ISIS started life as a research project at Cornell University but later become a commercial product distributed by Isis Distributed Systems, Inc. now a wholly owned subsidiary of Stratus Computing Inc.[Stratus96]. ISIS has been used as the basis for inter-process communication in the components of the DAQ including the run-control system. For an overview of ISIS see [Jones92].

3.5.1 Run-Control

As with the State Manager of the MODEL project, it was decided to model the behaviour of the various components in terms of finite state machines. Hence the run-control system provided a way of defining and commands for interrogating and manipulating finite-state automata. An application defined its state machine in a separate finite state machine (FSM) file. The FSM file was translated in to a C module using a custom-made LEX and YACC based translator. The C module was then compiled and linked with the application's code.

A state machine consisted of a list of states and commands that could be accepted in each state. Each command consisted of a command name, the name of the new state to which it would transfer if the command was executed successfully and the name of a user routine to be called to perform the processing associated with the command:

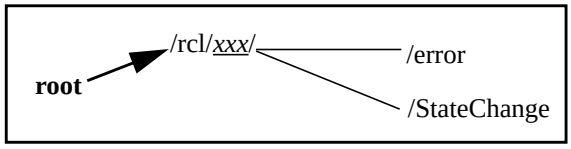
```
command -> new_state (user routine)
```

The run-control facility (rcl) validated each request to execute a command based on these definitions. Commands were executed when received by the target application. They were also displayed in the command list of the rclCmd graphical user interface program (Figure 8).

3.5.1.1 Hierarchy of Run-Control Programs

Given the size and distributed nature of the DAQ, it was decided to use a hierarchy of run-control programs that co-operated to subdivide the domains of the experiment. At the top of the hierarchy was the run-control program representing the state of the DAQ. It interacted with sub-run-control programs that were responsible for domains such as read-out or recording. The advantage of such a hierarchy was that the programming task could be broken down in to smaller subtasks. The top-level run-control program could be written in terms of the states provided by the sub-run-control programs which in turn could handle the complexities of the individual applications under their control. The hierarchy of run-control programs was mapped onto ISIS process groups as shown in Figure 7.

Figure 7 run-control ISIS process group hierarchy



The process group, *rcl*, acted as the root to which all applications became members. The separation into subgroups *xxx* allowed multiple copies of the run-control component to exist on a machine at the same time and support partitioned DAQ systems. The *State-Change* subgroup was used to broadcast changes of state of applications. The *error* sub-group was used by the Error Message Facility (EMF) [Ferrato93] to broadcast reported error messages. EMF offered a service equivalent to that provided by the EMU component of the MODEL project (see the published paper entitled *MODEL: A Software Suite For Data Acquisition* on page 21).

3.5.1.2 Implementation

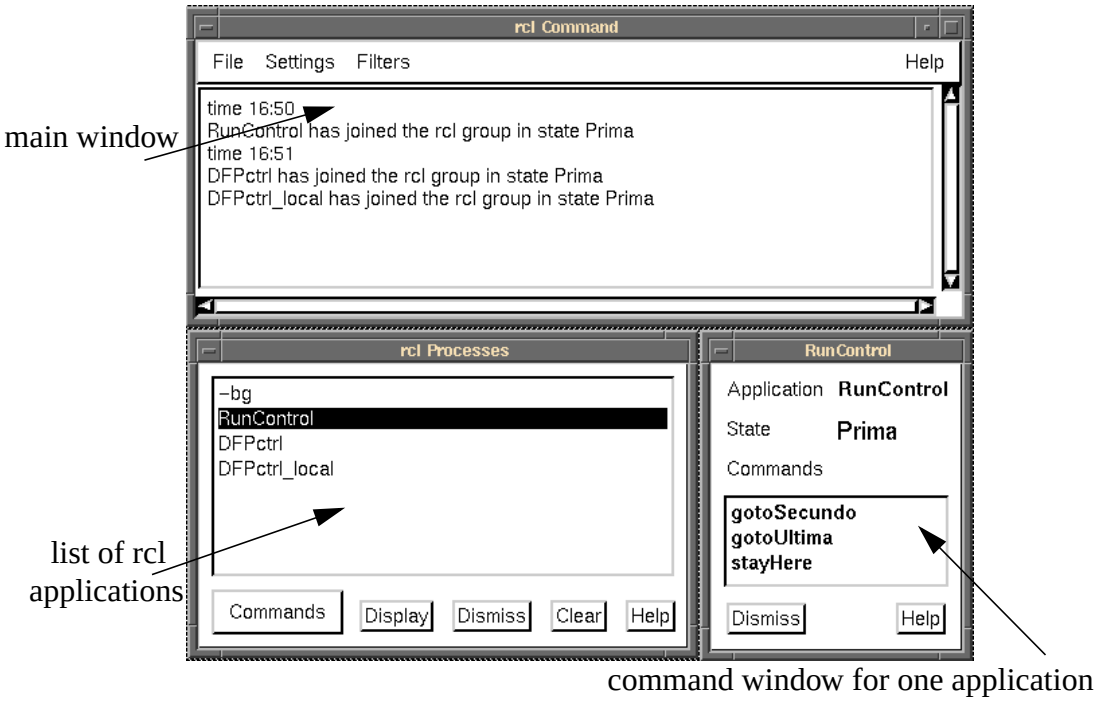
The facilities offered by the run-control and error message facility to application programs were packaged as a library containing a number of routines that could be invoked

by the application via a C language programming interface. All inter-process communication and multi-tasking facilities were provided by ISIS.

3.5.2 User Interface

The rclCmd user interface (Figure 8) provided a means of monitoring and sending commands to all controllers in the run-control facility. It was independent of the DAQ configuration and a tool for investigating activity at the ISIS process-group level. It was possible to run multiple copies of the user interface concurrently.

Figure 8 rclCmd user interface



The following published paper entitled *Building Distributed Run-Control in UNIX* on page 88 provides an overview of the ISIS package and how it has been used in the RD13 DAQ system.

Published paper

Building Distributed Run-Control in UNIX

Computing for High Energy Physics Conference 1992, Annecy, France ISBN 92-9083-049-2.

Published paper

Published paper

Published paper

3.6 Conclusions on the Use of ISIS for Inter-Process Communication

The adoption of ISIS overcame many of the limitations of the OSP package developed in the MODEL project (see *Conclusions on OSP* on page 53.), including:

- the possibility to run multiple co-operating servers avoided a single point of failure and allowed scaling with the size of the experiment,
- broadcast communication was used where supported by the underlying TCP/IP implementation,
- portability to more platforms (ISIS runs on most UNIXs, VMS and the author ported it to the EP/LX real-time operating system used on the front-end processors),
- support for all basic data types as well as user defined structures,
- support for various styles of communication (see *Communication Styles* on page 55).

However, ISIS did have a few limitations of its own:

- the programming interface was quite low-level, requiring the programmer to pack and unpack variables to and from messages,
- the thread facility provided by ISIS, while useful, did not use the native kernel threads on all platforms and hence a single thread could block the whole process when performing synchronous I/O.

3.6.1 Conclusions on the RD13 Run-Control Component

The RD13 run-control component offered a number of advantages when compared to MODEL's State Manager facility:

- the number and organisation of individual run-control programs could be defined in a configuration data store and no re-compilation or re-linking was necessary to modify a configuration,
- the integration of user written code for transition actions was simpler since they were all implemented in the same language and could be combined in the same thread or process,

- the rclCmd graphical user interface (Figure 8) was independent of the configuration, the controllers' FSMs and more flexible than the State Manager interface developed during the MODEL project (see *Graphical User Interface for the MODEL State Manager* on page 57),
- its implementation was shown to be more portable,
- individual controllers could be distributed across the network rather than all held within a single process.

However, the first version did have the following limitations:

- to make the FSMs inter-dependent it was necessary to write C code to monitor state changes and components starting or stopping. This was a low-level way of expressing these dependencies,
- since the FSM was statically linked with the component it could not be changed according to the DAQ configuration used,
- only one action (a call to a user routine) could be performed when a command was received. Multiple actions including sending commands to other components (without coding this in C) would have been more useful,
- it was assumed each command caused a state change - even if the new state was the same as the current state,
- actions could not be executed asynchronously.

3.6.2 The Meta Toolkit

In an attempt to simplify the definition of inter-dependencies of the controllers in the hierarchy, the author investigated the use of the Meta [Marzullo91] product built on top of ISIS. Meta and this investigation are described in the published paper entitled *Building Distributed Run-Control in UNIX* on page 88. Another paper [Jones92] explaining the rationale for this work was published during the initial evaluation of Meta. In architecture and concept terms, Meta was shown to be suitable for the development of the run-control system. Retrieving information about the state of the DAQ via sensors and modifying it via actuators allowed one to concentrate on the domain specific issues of the system. It also addressed some shortcomings of the ISIS package (see *Conclusions on the Use of ISIS for Inter-Process Communication* on page 93). Unfortunately, the

implementation of Meta was not as reliable or robust as the underlying ISIS toolkit. The distributed rule interpreter was particularly fragile. As a consequence, the author decided to add some of the features of Meta to the existing ISIS based run-control system. In particular, the author wanted to make use of the sensor/actuator concept. In the ISIS based run-control, events that caused state transitions to fire could be considered equivalent to sensors and user defined action routines to be actuators. With this in mind, the author extended the types of events sensed by the run-control via the FSM definition language. This work is described in the following chapter on the upgraded RD13 DAQ system.

3.7 Configuration Data Storage

Four distinct data stores were envisaged for the RD13 DAQ to store configuration information: hardware configuration (described the layout of the hardware in terms of crates, modules, processors and interconnects); software configuration (described the layout of the software in terms of processes, services provided, connections and host machines); run parameters (e.g. run number, recording device, level 2 trigger state etc.) and detector parameters (information pertaining to the detector and defined by the detector group themselves within a fixed framework).

3.7.1 First Implementation: StP/ORACLE

The first implementation of the configuration data storage framework was made using “traditional” (i.e. for which extensive expertise existed at CERN) technology: the entity-relation (E-R) data model and relational database management (DBMS) technology (as implemented by ORACLE). The E-R data model had already been used in several experiments [Green89] and was shown to be adequate for the purpose. The Software Through Pictures (StP) CASE tool supporting structured analysis and design (SA/SD) techniques was used to design and develop the data model for the DAQ’s software configuration. StP provided a graphical E-R editor capable of producing SQL code to create ORACLE (and other relational DBMS) tables and C data structures defining entities and relationships.

3.7.1.1 Real-Time Facilities and Distributed Access

ORACLE was not suited for real-time distributed access to the data [Mornacchi92] and its interfaces were not available for the front-end processors. In order to use ORACLE,

for configuration data storage in the DAQ an architecture needed to be devised that overcome this limitation.

For real-time access, a two-level scheme was adopted: an off-line level where the data was created and maintained in ORACLE (creation, insertion, updates, etc.); and an on-line level where, at run-time, the contents were extracted from ORACLE and read into an application's memory (i.e. ORACLE tables mapped onto arrays of data structures representing database rows). The data was accessed by DAQ programs via a user defined and implemented Data Access Library (DAL). The DAL provided an interface to the queries and updates needed by the applications. The DAL hid the actual implementation of the data store from the application as well as the detailed data model. This implied that the underlying data storage system could be replaced without affecting the application programs.

3.7.1.2 User Interface

Graphical tools were needed to allow physicists to browse and update the data. Such interactive programs performed all the operations allowed at run-time via the DAL. Generic browsing and updates, assumed to be rare and performed by experts only, were made using the commercial DBMS query language and facilities.

3.7.1.3 Intermediate File

The DAL navigated the in-memory tables to access the data. Initial loading of the database was relatively slow. To provide access to the data from the front-end processors the data was extracted from the ORACLE database and read into a workstation disk file (NFS being available throughout the system). Applications running on the front-end processors could read the disk file to fill their in-memory tables.

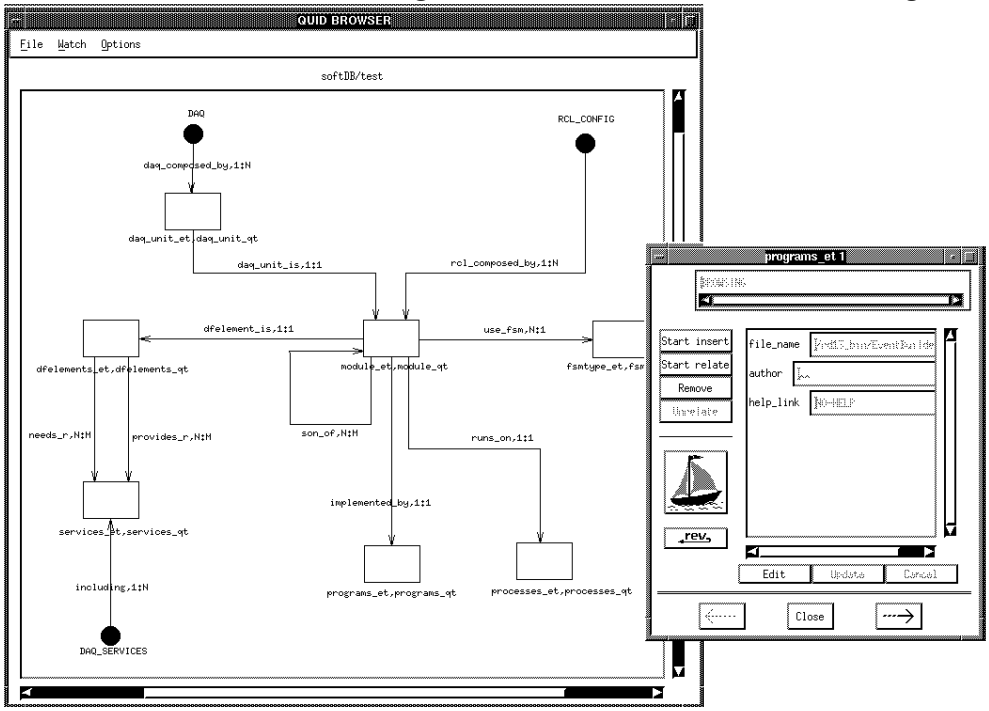
A major drawback of this scheme was the need to hand code the DAL directly in C, particularly the navigation of the data schema.

3.7.2 Second Implementation: QUID

An alternative to commercial relational DBMS are in-memory bank systems. They do not have all the functionality of relational DBMS but do not have so many overheads in terms of performance and demands on the underlying operating system. One such com-

mercial in-memory system is QUID (Artis S.r.l, Italy.) QUID is a data store development environment targeted to real-time applications (i.e. where performance is needed and the full functionality of a DBMS is not). An overview of QUID is included in the published paper entitled *Software Engineering Techniques and CASE Tools in RD13* on page 99. Figure 9 shows the graphical schema editor data browsing tool provided with the QUID environment.

Figure 9 QUID editor and browser showing one view of the RD13 DAQ software configuration



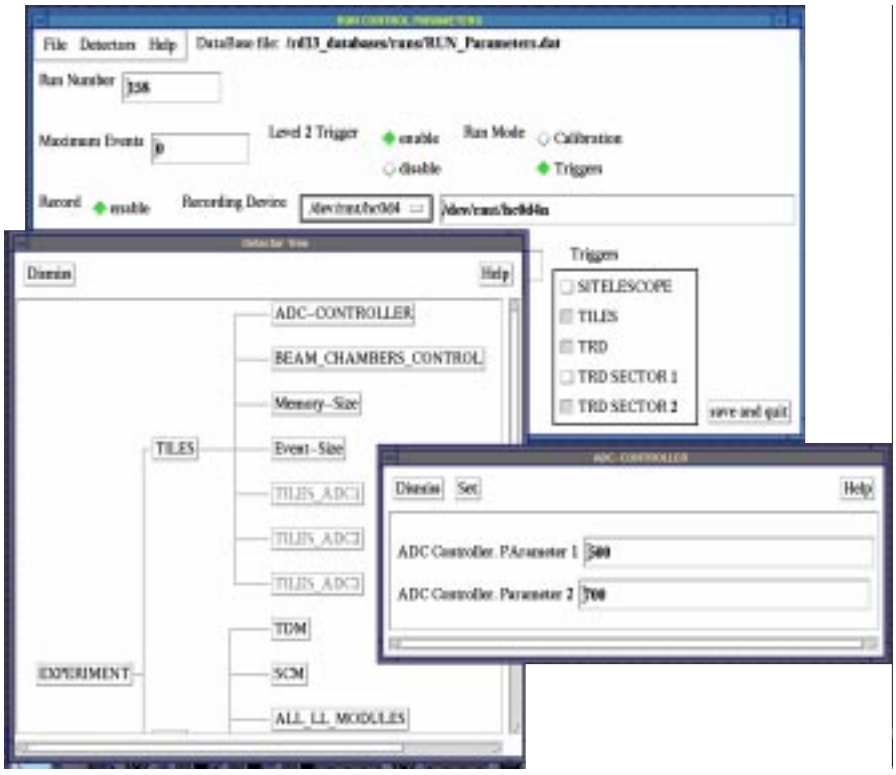
QUID allows the modelling, storing and handling of the data but it is not a full database management system since it relies on the host's file system for storing data and does not provide a multi-user environment. There is no provision for distributed transactions, nor for the distribution of data among many processes. For read-only data stores this restriction is easily overcome by having all participating processes access the disk files via NFS. When writing is also needed, a special scheme has to be devised. It also has some other important deficiencies:

- all the data is in the memory of the application which sets strict limitations on its size,
- no referential integrity or concurrency control is provided,

- no schema evolution facilities are provided. If a schema change takes place then the data that correspond to the old schema are no longer valid.

However, QUID was successfully used to implement all four RD13 DAQ configuration data stores mentioned earlier. Figure 10 shows the custom-made Motif based user interface to the detector parameters data store. This situation was satisfactory for most purposes but the arrival of object database management systems (ODBMS) made it possible to extend the functionality of the data stores. Other groups have also investigated the use of ODBMS systems in other HEP applications such as physics event recording [RD4596] [PASS94] [Le Goff95].

Figure 10 User interface to detector parameters data store



The migration from ORACLE to QUID data stores was simplified by the use of the DAL that allowed applications to be ported by simple recompilation. QUID's query language was found to be easier to use and simpler to integrate than ORACLE SQL.

The following published paper, entitled *Software Engineering Techniques and CASE Tools in RD13* on page 99 summarises the state of the graphical user interfaces and configuration data storage in the RD13 DAQ system at the end of the project's first phase. It also proposes areas of research for each of the software technologies for the second phase (described in the next chapter).

published paper

Software Engineering Techniques and CASE Tools in RD13

Third International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics (AIHEP'93) conference, Oberammergau in October 1993. ISBN 981-02-1699-8.

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

3.8 Conclusions on RD13 DAQ

The first phase of the RD13 DAQ project successfully demonstrated that MIPS based processors running a full multi-tasking real-time UNIX operating system in an VME environment could accommodate LHC scale data rates. The project also showed that UNIX was a suitable operating system for back-end DAQ activities and as a software development environment. The conclusions on the use of ISIS for inter-process communication have already been stated (*Conclusions on the Use of ISIS for Inter-Process Communication* on page 93).

3.8.1 Graphical User Interface Toolkits

The adoption of the X Window System and OSF's Motif widget set provided adequate support for basic graphical applications on multiple UNIX platforms in a local area network. This combination allowed the replacement of custom-made packages, such as the MHI package from the MODEL project, with commercial solutions. Business style graphics widgets (such as Bell Corps. matrix widget or the DataViews kit) could be used alongside Motif to extend the graphics facilities. The use of a graphical user interface builder CASE tool simplified the development of interfaces and greatly reduced the amount of hand-written software.

3.8.2 Configuration Data Storage Techniques

By adopting a common approach to configuration data storage on the RD13 DAQ it was possible to share information more easily between the various software components. The use of the entity-relationship approach, supported by ORACLE, provided the possibility to specify the configuration data required using a well defined data model. By replacing ORACLE with QUID, it was possible to provide easier access to the data store from the front-end processors and improve the run-time performance of the whole system. QUID's graphical schema editor and browser provided a convenient means for developers to design and populate the data stores but custom-made interfaces were required for the DAQ operators. As the size of the DAQ system grew and modifications were made to its structure, it became evident that the data size restrictions of QUID and its lack of support for schema evolution would become a problem in the future. This situation encouraged the investigation of alternative techniques and led to the evaluations of the object DBMS as described in the next chapter.

chapter

The RD13 Data Acquisition System Upgrade

4 The RD13 Data Acquisition System Upgrade

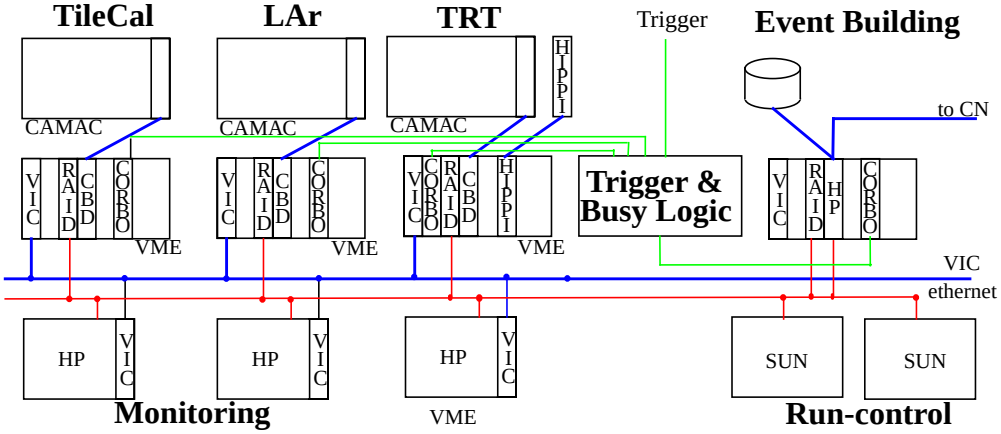
4.1 Introduction

This chapter describes the studies performed on the software technologies within the context of the third DAQ system. This DAQ system was an upgrade to the original RD13 DAQ, described in the previous chapter, and covers the period from January 1994 to December 1995. A published paper defines the improvements made to the DAQ system in this phase and is followed by a paper dedicated to configuration data storage techniques. The third and final paper of this chapter gives the status of the three software technologies at the end of the RD13 project.

4.2 RD13 DAQ Upgrade Overview

In order to cope with increasing requirements from the detectors and to further explore the scalability issues of the read-out system, the RD13 DAQ was upgraded to cope with multiple concurrent detectors. This provided the opportunity to upgrade some of the software components [Mapelli94]. Upgrades were made to the inter-process communication system used by the run-control, various graphical user interfaces and alternative configuration data storage techniques were explored. A combined run of several ATLAS sub-detector prototypes including the tile calorimeter (TileCal), liquid argon (LAr) calorimeter and transition radiation tracker (TRT) detectors in April 1996 demonstrated the successful use of the upgraded hardware and software. This version of the system [Mapelli95] was used in the H8 test beam area by ATLAS sub-detectors throughout 1996 (Figure 11).

Figure 11 DAQ Hardware setup at ATLAS test-beam 1996



The following published paper entitled *The RD13 Data Acquisition System* on page 113 describes the modifications made to the RD13 DAQ system during its upgrade.

Published paper

The RD13 Data Acquisition System

Computing for High-Energy Physics Conference 1994,.San Francisco, USA, Proceedings of CHEP94. Lawrence Berkeley Laboratory LBL-35822; CONF-940492; UC-405

Published paper

Published paper

Published paper

Published paper

Published paper

4.3 Configuration Data Storage

The QUID data store had been successfully used to hold DAQ configuration information in the original RD13 DAQ system. A number of limitations of the QUID system had been identified, as described in the following published paper entitled *Experience Using a Distributed Object Oriented DataBase for a DAQ System* on page 121, namely:

- the whole data store was loaded into the application's memory at initialization thus limiting the size of the store to that of the virtual memory assigned to a single process,
- no facilities were provided for referential integrity, concurrency control, schema evolution and versioning.

To overcome these limitations, investigations of the possible use of a commercial ODBMS in 1993 were made, as mentioned in the published paper entitled *Software Engineering Techniques and CASE Tools in RD13* on page 99. The initial investigations were based on Ontos [Ontos92] but it was soon abandoned because the version available at the time did not support heterogeneous client and server configurations. As an alternative, the hardware configuration data store and an associated application [Ambrosini94] were re-implemented using the GemStone [GemStone92] database. GemStone client libraries were not available for the operating system on the front-end processors and so the database could not used by applications running on these machines.

A third commercial ODBMS, called ITASCA [Itasca92], was available at this time and the company was prepared to port their client library to the front-end processors for testing purposes. ITASCA was a commercial extension of the ORION-2 research prototype from MCC (Microelectronics and Computer Technology Corporation).

Given that the client libraries were available on the front-end processors it was interesting to understand how far the database could be integrated into the DAQ system. The DAQ was modified to access the database in the data flow protocol transporting the physics data. This provided measurements of the delay introduced by making a database access on a per event basis. The results showed a minor degradation in the event rate when the client-side object caching capability of ITASCA was used. Investigations into

the use of commercial ODBMS were continued in the ATLAS experiment as described in the next chapter.

As an alternative to ODBMS, the published paper entitled *Applications of an OO Methodology and CASE to a DAQ System* on page 136 describes investigations into the use of the Kappa persistent object manager for configuration data storage purposes. Kappa's internal object persistence was used in the Error Message Facility (EMF) server and on-line book-keeping applications. A single function call was needed to save or restore a set of objects to or from disk. When used in this manner, the Object Diagrammer of the CASE tool became a graphical database schema editor. The system also provided limited schema evolution allowing objects to be saved to disk, the schema changed and the database re-read into memory. The facilities of the object persistence could not be compared to a real database since it had no notion of transaction support, concurrency control, versioning or distributed access but it was more akin to an object-oriented version of QUID.

Published paper

Experience Using a Distributed Object Oriented DataBase for a DAQ System

CHEP'95: Proceedings of the Internal Conference on Computing for High Energy Physics'95

18-22 September 1995. Rio de Janeiro, Brazil. ISBN 981-02-2783-3.

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

4.4 Inter-Process Communication

The principle use of the inter-process communication system in the RD13 system was the run-control system and error message facility as described in the previous chapter (*Inter-Process Communication* on page 85). The following revisions were made to the run-control system taking into account the lessons learned from the investigations performed with the Meta toolkit (see *The Meta Toolkit* on page 94).

4.4.1 Extended the Run-Control Transition Types

The finite state machine declaration notation (See “Run-Control” on page 85) was revised to incorporate the concept of sensors as defined in the Meta toolkit by adding more types of transitions:

- *Error signal* transitions executed when the corresponding EMF error message was reported by any rcl application:

```
errorCode -> new_state (user routine)
```

- *Rcl application started* transitions executed when the given application (identified by name or matching regular expression) connects to rcl:

```
appName started -> new_state (user routine)
```

- *Rcl application stopped* transitions executed when the given application (identified by name or matching regular expression) disconnected from rcl or exited:

```
appName stopped -> new_state (user routine)
```

- *Rcl application state change* transitions executed when the given application (identified by name or matching regular expression) entered the given state (identified by name or matching regular expression):

```
appName in some_state -> new_state (user routine)
```

When the new transition types were combined with the ability to use regular expressions to identify the application in the start, stop and state change transitions, this brought important improvements to the run-control component. When combined with naming conventions for application names, regular expressions made FSMs independent of the DAQ system configuration.

For example, assuming the top-level controllers for each detector were named *Controller_<detector name>* then a single transition could react to a state change of any detector:

```
Controller_* in Error -> Error(handle_detector_error)
```

4.4.2 Configurable Finite State Machines

By storing the definition of the FSMs in a data store and retrieving them at run-time, it became possible to select the appropriate FSMs according to the DAQ configuration being used. Once a FSM had been loaded from the database, it was put in the initial state and executed as before.

4.4.3 User Interface

The initial run-control user interface, rclCmd (see *User Interface* on page 96), was provided as a means of interacting with any controller in the hierarchy. A higher-level of abstraction was more appropriate for the DAQ operators that restricted interaction to the root of the control hierarchy and encompassed facilities for interacting with other parts of the DAQ. The DAQ Main Window (DMW) (Figure 12) was thus defined and provided:

- selection of system configurations,
- initial DAQ startup and state change push-buttons,
- run state information (updated dynamically) and
- access to all DAQ utilities (e.g. status display, monitoring tasks, etc.) via pull-down menus.

DMW integrated several commercial packages including Motif graphics, ISIS and the QUID data store. This resulted in a number of integration problems since each tool promoted a different style for steering and organising programs. A compromise was found limiting the use of a tool’s functionality to that which could coexist with other products. DMW was driven by the contents of the QUID data store and was hence independent of the DAQ system configurations. Given its on-line requirements (i.e. it should respond before the user becomes impatient) such a module would have benefited from maintaining the DAQ status in a distributed data management system.

Figure 12 DAQ Main Window user interface

4.4.4 Run-Control Issues

The measures described above improved the run-control system but a number of issues concerning the run-control had not been investigated or needed re-assessment:

- a better means of indicating to the operator the state of the run-control component itself. The time taken to execute a transition was variable and it was sometimes difficult to know if the DAQ system was stable, trying to execute a command or engaged in an automatically triggered transition,
- assess the relative merits of having the controller state coherency management distributed, as it is in the RD13 DAQ, or centralized as it was in MODEL's State Manager.

4.4.5 The Resource Manager

In distributed DAQ systems it is necessary to start and stop a plenitude of programs during the initialization and shut-down phases on many processors within the LAN. In the MODEL project, which was essentially running on a VAX/VMS cluster, the cluster management facilities provided by the VMS operating system could be used to fulfil this task and were encapsulated inside a dedicated component called MPC [Matheys89].

However, no such distributed process management facilities exist in the UNIX operating systems that hosted the RD13 project. Initially the UNIX's *rsh* facility was used but it satisfied only a subset of the requirements. In particular, it had a poor interface and little support for error handling. As an alternative the use of the Resource Manager, a toolkit built on top of ISIS, was investigated as described in the published paper entitled *Building Distributed Run-Control in UNIX* on page 88. The Resource Manager was intended to provide job control and load-balancing services to sets of workstations. Many alternative products existed both commercially and as freeware (e.g. Unison's Load Balancer [Unison97] or NASA's NQS [Kaplan93]). The Resource Manager could start processes but it lacked the necessary hooks to inform clients when a process died and would instead try to reschedule the job on another host. Such products emphasise maintaining a maximum through-put of jobs. In a DAQ system, the termination of an individual process could have important consequences in components such as the run-control that could not be resolved by simply starting another copy. As a result, the RD13 system reverted to the use of rsh but the issue is addressed again in the ATLAS prototype DAQ described in the next chapter.

4.4.6 Artifex

An alternative means of designing and implementing the run-control system was explored during the revision of the RD13 DAQ system. As described in the published paper entitled *Software Engineering Techniques and CASE Tools in RD13* on page 99, a Petri Net based approach had been employed in the design and implementation of the Data-Flow Protocol (DFP) component of the DAQ. This approach used PROTOB [Bruno95], an object-oriented method based on an extended data flow model defined using high level Petri Nets supported by a CASE toolset called Artifex (Artis S.r.l., Italy.) Artifex consisted of several tools supporting specification, modelling, prototyping and code generation activities within the framework of the software life cycle (system analysis, design, simulation and implementation):

- the analysis and design phases were made using the graphical formal language for the high level concurrent Petri Nets,
- during the simulation, generation and execution phases the user could simulate, set break points and step through the concurrent task model,

- the emulation supported distributed code generation from the same model used for analysis and simulation. During emulation, visual monitoring of the Petri Net was possible using the same GUI as for the previous phases.

This work showed that although the performance of the resulting system was not satisfactory for the processing of physics data [Khodabandeh93] it was adequate for other components such as the run-control system. The Artifex toolkit offered many of the facilities required for the development of the run-control system, including a means of modelling dynamic behaviour in terms of Petri Nets and a transparent network based communication between different parts of the model. The work on the DFP component [Fumagalli93] had shown the advantages of an integrated design and implementation environment with tools to simulate the model.

Hence a prototype of the RD13 run-control system was made with Artifex and a summary of the results is included in the published paper entitled *The RD13 Data Acquisition System* on page 113 and further detailed in [Aguer94]. Unfortunately, some problems were encountered during the development:

- integrating the code generated by Artifex with hand-written or third party code was cumbersome [Artifex93]. In particular, combining the event loops required by ISIS and the X Window System with that of Artifex required some programming gymnastics that would be better avoided [Jones93],
- Artifex did not support dynamic configuration. Every instance that participated in the model had to be explicitly drawn with the GUI and connected to other instances during the modelling phase. This was acceptable during the initial modelling and validation, but it became a restriction in the final application where a degree of dynamism was necessary [Jones94],
- the developers found modelling controllers with Petri Nets less obvious than with finite state machines.

Based on the above points it was decided to abandon the use of Artifex for run-control purposes. Nevertheless, the work with Artifex showed the benefit of using a transparent network communication facility and using a simulator to test models before full code generation as well as a CASE toolset to support most of the software lifecycle. These principles were carried through to the work involving the Object Management Work-

bench (OMW) described below. The better support for the software lifecycle that code generation provided was appreciated since it became possible to keep the design and implementation in step (i.e. the code is generated from the design) over successive releases of the software.

4.4.7 Kappa CommManager

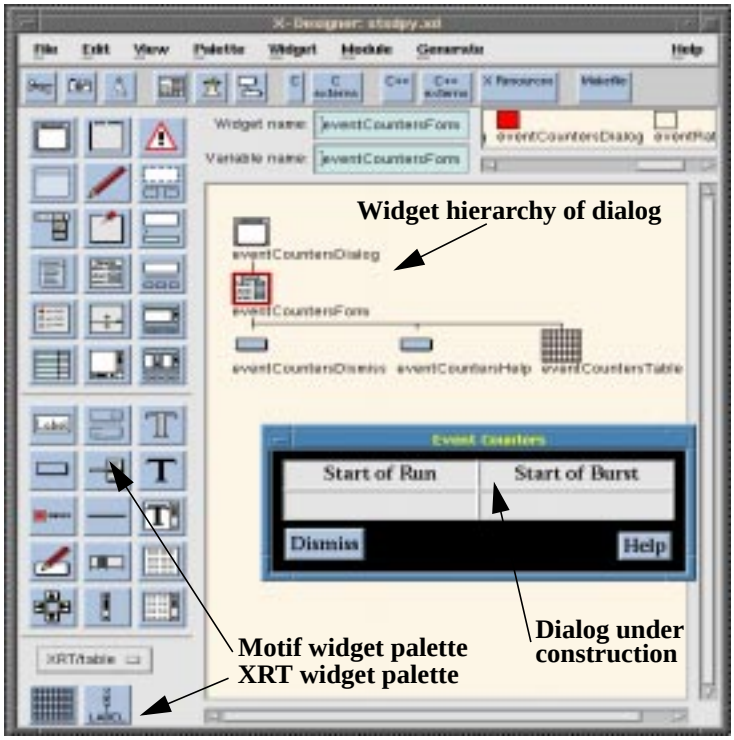
As an alternative to ISIS and Artifex's transparent network communication, the published paper entitled *Applications of an OO Methodology and CASE to a DAQ System* on page 136 describes the use of the OMW integrated CASE toolset that included an inter-process communication package called the Kappa CommManager. OMW applications could be distributed between UNIX workstations and PCs running Microsoft Windows by using the Kappa CommManager [Intellicorp93]. Kappa CommManager provided transparent inter-process communication among distributed objects running over TCP/IP networks and complied to the CORBA protocols defined by the Object Management Group [OMG95]. Limited tests with the CommManager package for distributed applications were performed but it was not possible to incorporate its use in the DAQ. It worked satisfactorily on a network of Sun workstations but it was not possible to test it in a heterogeneous environment. The programming overhead of distributing an application over several processes was minimal. Conceptually, it could be seen as an object-oriented equivalent of remote procedure calls (RPCs) with the advantage than the developer is not required to define the interfaces via a specialised language (e.g. as used by RPC compilers). As with RPCs, an application could be developed as a single process and distributed later with the minimum of disruption. Interest focused on combining the CommManager with the object persistence and object monitors in order to provide a distributed, reactive data store, whereby clients of a data server process could use monitors to be informed of any changes to the objects managed by the server.

4.5 Graphical User Interface Toolkits

In the original RD13 DAQ system, the Data Views graphics library and editor were used to develop sophisticated user interfaces such as the status display (Figure 4) and later the Data Views widgets were integrated in the X-Designer GUI builder. During the revision of the RD13 DAQ, the Data Views widgets were replaced with an alternative widget set called XRT sold by the KL Group [KL94]. These widgets offered more

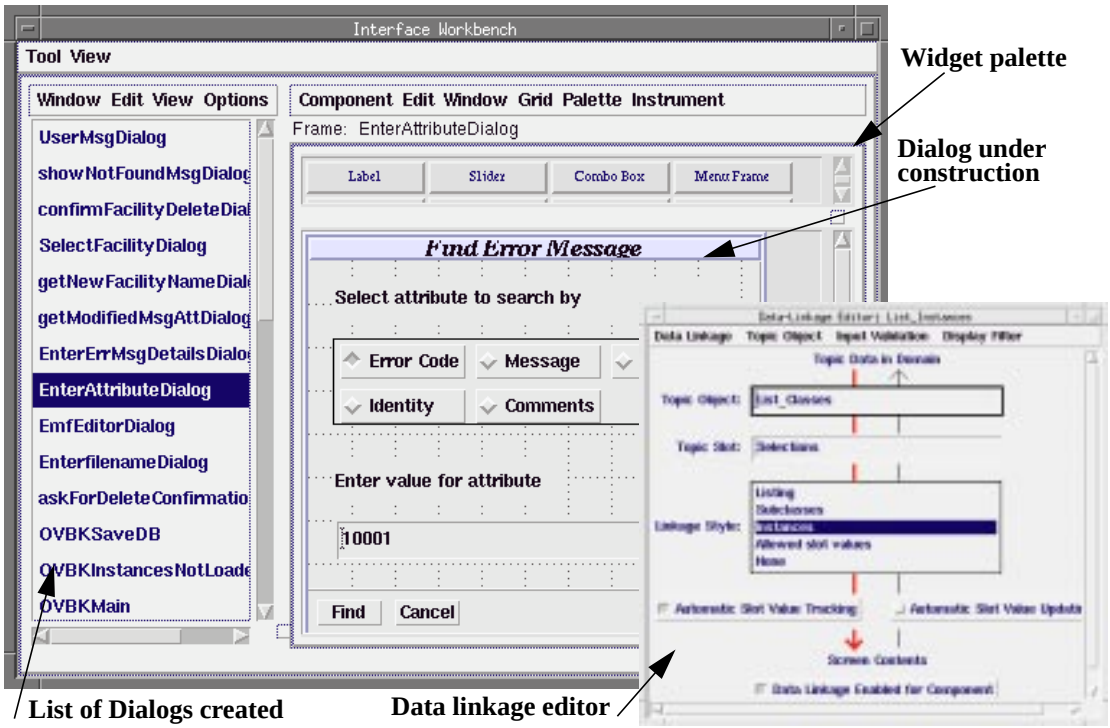
sophisticated graphics capabilities with a simpler programming interface and could also be integrated with the X-Designer GUI builder (Figure 13).

Figure 13 X-Designer graphical user interface builder with Motif and XRT widgets



For the components of the DAQ developed with OMW, the integrated GUI builder called the Interface Workbench was used (Figure 14). This tool provided similar functionality to X-Designer but also rectified an important short-coming with the GUI development technique at that point, namely the lack of support for modelling the relationship between the application and the interface. However, it was found that the integration of third party widgets was more complicated in the Interface Workbench than with X-Designer. A comparison between the use of X-Designer and OMW's Interface Workbench in included in [Jones95].

Figure 14 OMW's Interface Workbench and Data-Linkage Editor



The following published paper entitled *Applications of an OO Methodology and CASE to a DAQ System* on page 136 gives an overview of the OMW CASE toolset and describes how it was applied to various components of the RD13 DAQ system.

Published paper

Applications of an OO Methodology and CASE to a DAQ System

CHEP'95: Proceedings of the Internal Conference on Computing for High Energy Physics'95

18-22 September 1995. Rio de Janeiro, Brazil. ISBN 981-02-2783-3.

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

3.9 Conclusions on the RD13 DAQ Upgrade

The RD13 DAQ was successfully upgraded to support data taking from multiple concurrent detectors and to host switch based event building studies. The DAQ was used in test beams with several prototype ATLAS detectors and the project later became the basis of the ATLAS DAQ system.

3.9.1 Configuration Data Storage

In order to overcome the limitations of the QUID system three successive ODBMS (Ontos, GemStone and ITASCA) were evaluated as well as a persistent object manager (Kappa object manager). The use of the ODBMS showed the importance of the installed configuration on the performance of the applications. The ODBMS provided far more control on the consistency and integrity of the data but implied a greater management load. Their use of client-server based architectures solved the problem of simultaneous distributed access to a consistent set of data by multiple applications but introduced restrictions on how the applications could be implemented and was the source of integration problems with other software packages.

The Kappa object manager was similar to QUID but also provided limited schema evolution facilities. The OMW CASE toolset provided better support for the development of specialised graphical user interfaces for the DAQ operators.

3.9.2 Inter-Process Communication

ISIS continued to be the basis of all communication used in the DAQ control and configuration components when the RD13 system was upgraded. Improvements were made to the run-control system by extending the finite state machine definition notation using more advanced ISIS features. Attempts were made to employ ISIS layered products such as the Meta toolkit and the Resource Manager but their use had to be abandoned due to limitations of their implementations.

Two alternatives to ISIS were investigated: Artifex and the Kappa CommManager. The Artifex development environment provided transparent inter-process communication to exchange Petri Net tokens but it proved too rigid for day-to-day use in a physics experiment. The Kappa CommManager provided first access to OMG's Corba based communication but its use was too closely linked to the Kappa programming environment

which was not available on all the platforms used by the DAQ. This work did encourage the investigation of other Corba implementations as described in the next chapter.

3.9.3 Graphical User Interface Toolkits

The combination of the X Window System and OSF's Motif toolkit with the X-Designer GUI builder established during the first phase of the RD13 project continued to be the basis of the interfaces in the upgraded DAQ. The DataViews business graphics widgets were replaced with an alternative set called XRT that provided a simpler programming interface and easier integration with X-Designer.

As an alternative to X-Designer, the OMW's Interface Workbench was used to develop interfaces to several DAQ components. This tool was more closely integrated with the rest of the development environment and improved the connection between the application and the graphical interface. By using the Data Linkage editor, it was no longer necessary to write code by hand for many callback routines such as those to display data in widgets. This tool could also generate code for interfaces using Microsoft's MFC running on PCs as well as OSF's Motif. The issue of the connection between the application and its graphical interface is further developed in the next chapter.

chapter

The ATLAS Data Acquisition System Prototype

5 The ATLAS Data Acquisition System Prototype

5.1 Introduction

This chapter presents the state of the ongoing studies performed on the three software technologies. These studies were made within the context of the fourth and final DAQ system, called ATLAS DAQ prototype -1, and covers the period from January 1996 to the summer of 1997. The first published paper gives an overview of the ATLAS prototype DAQ and is followed by a paper describing the most recent studies in each software technology.

5.2 ATLAS DAQ Prototype -1 Project

The goal of the ATLAS DAQ -1 project is to produce a fully functional prototype suitable for evaluating candidate technologies and architectures for the final DAQ system of the ATLAS experiment at CERN's LHC future accelerator. The prototype consists of a complete "vertical" slice of the ATLAS DAQ architecture, including all the hardware and software elements of the data flow, its control and monitoring and all the other elements of a complete on-line system, from the detector read-out to data recording. For further information on the ATLAS experiment see [ATLAS94] [ATLAS96].

5.3 Graphical User Interface Toolkits

At the end of the RD13 project, a satisfactory development environment had been established for graphical user interfaces. This environment consisted of the X Window System and OSF's Motif toolkit as the basic windowing packages. Two alternative means of producing interfaces had been employed: The X-Designer GUI builder and the Object Management Workbench's (OMW) Interface Builder.

The X-Designer GUI builder had been successfully used to model interfaces using both Motif and XRT widgets (including tables, 2-D graphs, pie charts and so on for more sophisticated applications) then generate portable C code. The only apparent restrictions were the lack of support for the relationship between the application and the GUI, and the non-availability of the XRT run-time library on the front-end processors.

OMW's Interface Builder had been successfully used to implement GUIs for various DAQ components as part of the integrated OMW development environment. With its Data Linkage editor, the Interface Builder was able to model the relationship between

the GUI and the application thereby reducing the effort required to integrate the two domains. While the development was simplified, the deployment was more restrictive since the OMW run-time library was available on fewer platforms than Motif and required more resources to run.

At this point, PCs running Microsoft Windows NT became suitable platforms for DAQ activities and their introduction put into question the validity of the choice of the X Window System and the Motif toolkit. As a result, an evaluation of the Java programming language and the Abstract Window Toolkit (AWT) was performed [Caprini96].

5.4 Inter-Process Communications

ISIS had been successfully used as the basis of inter-process communication in many DAQ components. However, it was necessary to port the ISIS toolkit to the front-end processors which was possible only because the product was distributed in source code form. The commercial distributors of ISIS were not convinced of the business argument for supporting such platforms and were considering distributing new versions of the product in binary format.

The use of Object Management Group's Corba communication system had been explored through the tests made with the Kappa CommManager as part of the OMW environment (see "Kappa CommManager" on page 133). At the same time, implementations of the Object Management Group's Corba standard were becoming widely available. As a consequence, evaluations of freeware implementations of two emerging standards namely Message Passing Interface (MPI) [Touchard96] and Corba [Kolos96] were performed.

5.5 Configuration Data Storage

QUID was still the most widely used configuration data storage technique in the RD13 DAQ but prototype components had been made using the Itasca object database management system and OMW's persistent object manager.

The following published paper entitled *The ATLAS DAQ and Event Filter Prototype "-1" Project* on page 151 gives an overview of the DAQ project in which the software technology studies were carried out for the ATLAS experiment.

Published paper

The ATLAS DAQ and Event Filter Prototype “-1” Project

CHEP’97: Proceedings of the Internal Conference on Computing for High Energy Physics’97,

7-11 April 1997. Berlin, Germany. (being printed).

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

5.6 Software Technologies in the ATLAS Prototype DAQ Project

The following published paper entitled *Software Technologies for a Prototype ATLAS DAQ* on page 160 describes the studies on inter-process communication, graphical user interface toolkits, and configuration data storage techniques within the ATLAS prototype DAQ project. It is the final published paper included in this thesis and represents the culmination of the investigation into the three software technologies at this point in time. The architecture and requirements referred to in the paper benefit from the experience gained in the MODEL and both subsequent RD13 DAQ systems described in the published papers of the preceding chapters. A general principle has been the consistent application of a single implementation for each technology throughout all components of the back-end DAQ. This approach simplifies the management of the DAQ software, permits the sharing of information between different components and eases their integration.

Published paper

Software Technologies for a Prototype ATLAS DAQ

CHEP'97: Proceedings of the Internal Conference on Computing for High Energy Physics'97,

7-11 April 1997. Berlin, Germany. (being printed).

Published paper

Published paper

Published paper

Published paper

Published paper

Published paper

chapter

Discussion and Conclusions

6 Discussion and Conclusions

This chapter provides a summary of the activities and developments for each of the three software technologies by tracing the work of this research programme through the four successive DAQ development projects and suggests possible areas for future work.

6.1 Graphical User Interfaces

Within the MODEL project, studies of graphical user interfaces started with the development by the author of various elements of the human interface package (MHI). MHI provided distributed windowing facilities to DAQ software in a local-area network on a variety of devices including character-cell terminals and bit-mapped workstation screens. MHI was designed and implemented at a time when bit-mapped workstation screens were just being introduced and no clear windowing graphics standard existed. It helped to bridge the gap between radically different device types and at the same time allowed graphical applications their first possibility to exploit processing power in a distributed environment.

MHI was ported to the then emerging X Window System that provided support for a greater number of vendors' devices and also replaced the RPC based communication system. The porting of MHI to X confirmed the advantages of MHI's basic architecture that combined a client-server model using RPC-based communication with device independence. During the port it was necessary to marry the event-driven structure of X applications with the discrete library application programming interface of MHI. The facilities provided by the layered packages (i.e. menus, panels and dialogues) were later supported by the most widely used commercial toolkits namely Microsoft Foundation Classes (MFC) and Open Software Foundation's (OSF) Motif built on top of X.

This port also provided initial access to high-level windowing toolkits and commercial graphical user interface builder tools used to implement an interface to the State Manager run-control component of the DAQ.

At conception of the RD13 project, an evaluation of several windowing toolkits (Motif, InterViews and OpenLook) was made that led to the adoption of OSF's Motif toolkit and the X-Designer GUI builder. This combination was used to implement many user interfaces for the RD13 DAQ including the data store editors and the run-control display. The advent of graphical interface builders for the most common toolkits (e.g. X-

Designer for Motif and Visual Basic for MFC) simplified the task of graphical interface development. But a commercial market for such GUI builders could exist only when a common standard for graphical toolkits had gained wide acceptance. This was not the case when MHI was designed and implemented. To satisfy more sophisticated graphical needs than could be met by Motif, the DataViews graphics package was evaluated and used to implement interfaces such as the DAQ status display.

The adoption of the X Window System and OSF's Motif widget set provided adequate support for basic graphical applications on multiple UNIX platforms in a local area network. This combination allowed the replacement of custom-made packages, such as the MHI package from the MODEL project, with commercial solutions. Business style graphics widgets (such as Bell Corps. matrix widget or the DataViews kit) could be used alongside Motif to extend the graphics facilities. The use of a graphical user interface builder CASE tool simplified the development of interfaces and greatly reduced the amount of hand-written software.

During the upgrade of the RD13 DAQ, the DataViews package was replaced with the XRT widget set that integrated better with Motif and the X-Designer GUI builder. Investigations into alternative development tools were made using Intellicorp's Interface Workbench that provided better support for application and graphics integration. It was used to implement interfaces to the Error Message Facility and the Run Bookkeeper components. This tool had the advantage of being more closely integrated with the rest of the development environment and improved the connection between the application and the graphical interface. By using the Data Linkage editor, it was no longer necessary to write code by hand for many callback routines such as those to display data in widgets. The tool could also generate code for interfaces using Microsoft's MFC running on PCs as well as OSF's Motif on UNIX.

In the ATLAS prototype DAQ project, X-Designer has been re-evaluated for cross-platform development purposes and Motif has been compared to the facilities provided by the Java AWT toolkit. It is likely that Motif and X-Designer will continue to be used in the immediate future and the use of Java and AWT will be restricted to applications that require access to information from a wide area (e.g. world-wide access to the run book-keeper information and possibly the DAQ status display).

6.2 Inter-Process Communication Systems

The study of inter-process communication systems started with the use of RPCs to support distributed user interfaces in the MHI package of the MODEL project. In order to satisfy the needs of certain DAQ applications, the Occurrence Signalling Package (OSP) offering publish/subscribe style interfaces, was developed and used as a basis of communication in the run-control component and many other areas. The architecture of the OSP package offered several advantages over simple point-to-point RPC style connections, namely:

- single network connection per client,
- single server architecture to avoid synchronization issues,
- partitioned name space for support of multiple concurrent data taking activities,
- group based communication implying no modifications were needed when the sender-receiver configuration changed,
- simple client-interface for diverse applications.

But this architecture introduced certain limitations, namely:

- a single server was a single point of failure,
- no provision was made for restoring information for existing connections when the server restarted,
- a single server limited the scalability of the system,
- broadcasting was simulated and hence the performance degraded as the number of clients increased,
- the implementation relied on message buffering and asynchronous I/O features of the underlying DECNET network protocol,
- all information was transferred in ASCII format.

Within HEP software, the OSP package introduced a new style of communication in distributed applications. The majority of HEP software used point-to-point RPC type communication but OSP opened the possibility to develop another range of applications that extended the functionality of DAQ systems. OSP permitted three distinct kinds of

interactions to occur among applications: request/reply, broadcast request/reply and publish/subscribe.

At the time of the RD13 project, the limitations of OSP's implementation became clear and the ISIS toolkit was evaluated as an alternative. ISIS was used in several components of the DAQ including the run-control and error message facility. The adoption of ISIS overcame many of the limitations of the OSP package developed in the MODEL project, namely:

- multiple cooperating servers avoided a single point of failure and could scale with the size of the experiment,
- true broadcast communication was used where supported by the underlying TCP/IP implementation,
- portability to more platforms,
- support for all basic data types as well as user defined structures,
- support for various styles of communication.

However, ISIS did have a few limitations of its own, namely:

- low-level programming interface,
- the thread facility did not use the native kernel threads on all platforms.

To simplify the run-control component, a layered product called Meta was evaluated that provided many interesting facilities but lacked robustness. The Resource Manager package was also evaluated for job control purposes.

When the RD13 DAQ was upgraded, an investigation of the Petri Net based Artifex toolset including inter-process communication facilities was made and a prototype of

the RD13 run-control system was produced. Unfortunately, some problems were encountered during the development, namely:

- integrating the code generated by Artifex with hand-written or third party code was cumbersome,
- Artifex did not support dynamic configuration,
- the developers found modelling controllers with Petri Nets less obvious than with finite state machines.

Based on the above points it was decided to abandon the use of Artifex for run-control purposes. Nevertheless, the work with Artifex did show the benefit of using a transparent network communication facility and having a simulator to test models before full code generation. The CASE toolset support for most of the software lifecycle was also appreciated. Its code generation facilities encouraged the investigation of Intellicorp's Object Management Workbench that included transparent inter-object communication via the Corba based CommManager. This evaluation lead to further investigation of Corba implementations within the ATLAS project and the adoption of Xerox Parc's ILU package.

6.3 Configuration Data Storage

The issue of configuration data storage was not addressed in a consistent manner by the components of the MODEL project but an investigation into the potential use of a commercial relational database was made. In the RD13 DAQ, an initial implementation was made using ORACLE with Software Thru Pictures as a modelling and SQL generation tool. ORACLE was not suited for real-time distributed access to the data and its interfaces were not available on the front-end processors. In order to use ORACLE, an architecture needed to be devised that avoided this limitation. For real-time access, a two-level scheme was adopted: an off-line level where the data was created and maintained in ORACLE (creation, insertion, updates, etc.); and an on-line level where, at run-time, the contents were extracted from ORACLE and read into an application's memory (ORACLE tables were mapped onto arrays of data structures representing database rows). A drawback of this scheme was the need to hand code the user-defined data access library (DAL) directly in C, especially the navigation of the data schema. This implementation proved inadequate for real-time access and was replaced by the QUID in-memory data manager.

QUID allowed the modelling, storing and handling of the data but it was not a full database management system since it relied on the host's file system for storing data and did not provide a multi-user environment. There was no provision for distributed transactions, nor for the distribution of data among many processes. For read-only data stores this restriction was overcome by having all participating processes access the disk files via NFS. When writing was also needed, a special scheme had to be devised. QUID also had some other important deficiencies:

- the whole data store was loaded into the application's memory at initialization thus limiting the size of the store to that of the virtual memory assigned to a single process,
- no facilities were provided for referential integrity, concurrency control, schema evolution and versioning.

The migration from ORACLE to QUID based data stores was simplified by the definition of the DAL that allowed applications to be ported by simple re-compilation. It was also found that QUID's query language was easier to use and simpler to integrate than ORACLE's SQL.

Later, when the data management facilities of QUID were surpassed, investigations were made of a number of ODBMS namely, Ontos, GemStone and Itasca. The ODBMS provided far more control on the consistency and integrity of the data but implied a greater management load. Their use of client-server based architectures solved the problem of simultaneous distributed access to a consistent set of data by multiple applications but introduced restrictions on how the applications could be implemented and was the source of integration problems with other software packages.

Intellicorp's persistent object manager was also evaluated and used to implement the data stores for the Error Message Facility and Run Book-keeper components. A single function call was needed to save or restore a set of objects to or from disk. When used in this manner, the Object Diagrammer of the CASE tool became a graphical database schema editor. The system also had the advantage of providing limited schema evolution - that is to say, objects could be saved to disk, the schema changed and the database re-read into memory. The facilities of the object persistence could not be compared to a real database since it had no notion of transaction support, concurrency control, ver-

sioning or distributed access but it was more akin to an object-oriented version of QUID.

This work led to the acceptance of a two-tier approach to configuration data storage in the ATLAS prototype DAQ, combining a persistent object manager with a full ODBMS. For the persistent object manager, Rogue Wave's Tools.h++ C++ class library was evaluated and used as the basis of the OKS facility. Objectivity has been evaluated and selected as the ODBMS in accordance with data storage activities of other groups within the experiment.

6.4 Contribution to Knowledge

This research programme has addressed three key software technologies for HEP DAQ systems, namely inter-process communication systems, data storage techniques and graphical user interface toolkits for control and configuration purposes. Advances in these areas were necessary to cope with the increasing size, distribution and complexity of DAQ systems. This work consisted of understanding the requirements in each area, conducting a survey of existing available software (commercial or otherwise) then selecting and applying the most appropriate choice. Due to the scale and complexity of HEP DAQ systems, the survey has often showed that suitable software packages were not readily available. This lack of suitable existing software packages has lead the author to develop several packages described in this thesis, including MHI, OSP and the RD13 run-control system. Eventually the custom-made packages have been replaced with more widely used software when it has become available. This confirms the uniquely demanding nature of HEP DAQ systems that require specialised software several years ahead of other application domains.

Many of the applications within DAQ systems, such as the run-control system, show that these three software technologies cannot be treated in isolation. The coexistence of software packages to support inter-process communication, data storage and graphical user interfaces in the same application, often running in embedded processors, places extra restrictions on the design and implementation of such packages.

6.4.1 Graphical User Interface Toolkits

MHI provided the first device independent distributed windowing facility to HEP DAQ systems. The investigation of the X Window System and Motif toolkit defined a clear

strategy for the migration of windowing applications from MHI and other custom-made graphical toolkits to a main-stream commercial alternative.

The author's evaluation of OpenLook, Motif and Interviews, and the resulting recommendation of Motif helped guide the HEP community towards the most prevalent graphical toolkit for UNIX workstations of the 1990s. The author's early experiments with graphical user interface builder CASE tools demonstrated their advantages in terms of design abstraction and developer productivity.

6.4.2 Inter-Process Communication Systems

OSP provided the first documented opportunity for HEP DAQ systems to use publish/subscribe style inter-process communication. This extended the functionality of DAQ systems to areas where the more traditional client-server approach of Remote Procedure Call systems was not applicable. The use of OSP in various DAQ systems has led to the development of second generation publish/subscribe facilities such as DELPHI's DIM [Gaspar93] and L3's CPC [WenausT89]. As a continuation of the investigations started with OSP, the author evaluated and developed the first documented application of a commercial publish/subscribe facility in HEP DAQ systems [Birman90]. The transition towards more higher-level communication systems (as shown by the author's continuing investigations into the use of Corba) has helped DAQ developers concentrate on the design issues of their applications rather than the programming aspects of their implementation. This succession of investigations into progressively more sophisticated inter-process communication systems has helped to bridge the gap between the front-end processors and back-end workstations of DAQ systems.

6.4.3 Configuration Data Storage Techniques

The author's investigations into configuration data storage techniques has helped to make HEP DAQ systems more flexible and independent of particular detector configurations. The work has extended the use of data stores to more parts of DAQ systems. The investigations into the specific technologies has helped to show the advantages of using firstly the entity-relationship model and more recently object-oriented models over ad hoc solutions. The investigations show the first documented application of commercial ODBMS to configuration data storage and has helped convince the HEP community of the benefits of ODBMS and commercial products in general.

In all the investigations concerning the three software technologies, the author has promoted the use of documented software engineering techniques and CASE tools. This approach has helped to advance the use of software engineering techniques and CASE tools in the development of DAQ systems and HEP software in general to the point that the filed is now considered important by the laboratory directorate.

6.4.4 Author's Personal Contribution

The author was directly responsible for the following aspects of the studies described in this thesis.

- MODEL

As one of the major contributors to the Model Human Interface package, the author was responsible for the window manager layer including its porting to the X Window System, the layered dialog package and the graphical user interface for the State Manager tool. The author was the sole contributor to the design and implementation of the original version of the Occurrence Signalling Package and the DECNET multi-client extension of the RPC server.

- RD13

The author personally evaluated the OpenLook, Motif and InterViews graphical toolkits and various GUI builders including VUIT and X-Designer. Using the Motif toolkit and X-Designer GUI builder, the author developed many user interfaces for the DAQ including the status display, graphical event dump, rclCmd run control interface and various browsers and editors for the data stores. As the person responsible for the original run control system, the author evaluated the ISIS toolkit, ported it to the EP/LX operating system and implemented the layered rcl package including the FSM language and translator. He also designed the Error Message Facility and evaluated the Meta product. For the data stores, the author developed several data access libraries on top of QUID.

- RD13 Upgrade

The author evaluated the Ontos ODBMS, the Object Management Workbench CASE tool and developed the error message data store, server and graphical editor. The author was responsible for the extension to the FSM language for the run control system and

the evaluation of the Artifex CASE tool for run control purposes. He re-designed and implemented the DAQ status display using the XRT widget set.

- ATLAS

In the on-going ATLAS Prototype -1 project the author is the coordinator for the group of developers working on the back-end software components. He made the initial feasibility studies for several of the selected packages including ILU, Objectivity, CHSM and the Tools.h++ class library.

6.5 Recommendations for Further Work

The increase in size and complexity of HEP DAQ systems is continuing as the physicists prepare for the LHC accelerator. As a consequence, the needs in terms of the software technologies discussed in this research programme will continue to grow and hence provide a fruitful area of research. Given the economical and social situation of the funding member states of laboratories such as CERN, there is a drive to use more commercial off the shelf (COTS) software with the intention of reducing costs of developing and maintaining custom-made software. Due to the nature of DAQ systems, they are likely to remain one of the most demanding application domains and hence the choice of suitable COTS software is expected to be limited. Taking into account the long timescales of HEP projects such as the experiments on the LHC accelerator, it is necessary to insulate the software systems from the vagaries of the commercial market. One of the most appropriate means of meeting this requirement is to adopt relevant international standards and select conforming software. However, it may be necessary to influence new standards to take into account specific HEP DAQ requirements. Currently there are two evolving standards that appear to be appropriate for the software technologies discussed in this thesis: The Object Database Standard [Cattell93] for data storage techniques and the Common Request Broker (Corba) [OMG95] for inter-process communication systems. Unfortunately, there appears to be no similar emerging standard for graphical user interfaces where the market is divided along operating system alliances. The facilities provide by Java and AWT appear promising, but it is too early to indicate if it will prove a lasting alternative.

The Object Database Standard is being tracked and actively influenced by the RD45 research project at CERN [RD4596]. However, the project’s goals are to provide persistent data storage for bulk physics data during the analysis and reconstruction phases

that are performed off-line. The use of ODBMS for DAQ configuration data storage is outside the scope of the RD45 project and this is one of the reasons that the ATLAS prototype DAQ project adopted a two-tier approach. However, there are many advantages (in terms of software and programmer portability) in using the same programming interface to the ODBMS and in-memory object manager that constitute the two tiers. The development of such an object manager and a means of migrating data to and from the ODBMS appear to be areas for future research.

For the ATLAS DAQ prototype, the Corba standard for inter-process communication has been adopted. Conceptually, it can be seen as the object-oriented equivalent of remote procedure calls (RPCs) and hence does not support the publish/subscribe facilities required for DAQ components such as the run-control system. However, in the recent Corba 2 standard a number of layered services have been introduced. One such service is the Event Service [OMG96] that provides publish/subscribe facilities. It is expected that the Event Service, along with other services defined by the standard, could be used to meet many of the communication needs within a DAQ system and thus their application could be areas for future research.

References

ATLAS94 . ATLAS Technical Proposal, The ATLAS Collaboration, CERN/LHCC/94-43, LHCC/P2, December 1994.

ATLAS96. ATLAS Computing Technical Proposal, The ATLAS Collaboration, CERN/ LHCC/96-42, 15 December 1996, ISBN 92-9083-092-1.

Ada83. Ada Joint Program Office, United States Department of Defence, Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, Washington D.C.: Government Printing Office, 1983.

Adam91. Architecture and Performance of the DELPHI Data Acquisiton and Control System. DELPHI DAQ Group, W. Adam et al., Computing for High-Energy Physics Conference 1991, Tsukuba, Japan. ISBN 4-946443-09-6.

Aguer94. Software Engineering Techniques and CASE Tools in RD13, Aguer M et al., 3rd International Conference on Accelerator and Large Experimental Physics Control Systems - ICALEPCS '93 Berlin, Germany ; 18 - 23 Oct 1993 . Publ. in: Proceedings, W Busse and M C Crowley-Milling Nucl. Instrum. Methods Phys. Res., A : 352 (1994) (383-385)

Ambrosini94. OODBMS for a DAQ system, G. Ambrosini et al., Proceedings of the Conference on Computing in High Energy Physics' 94. San francisco, USA, 21-27 April 1994. LBL-35822,CONF-940492, UC-405.

Angelov91. Performances of the central L3 Data Acqusition system, T. Angelov et al., Nuclear Instrments and Methods in Physics Research A306 (1991) 536-539, North-Holland.

Arnold96. The Java Programming Language, K.Arnold, J.Gosling, Addison-Wesley Publishing, ISBN: 0201634554 (1996).

Artifex93. Known Problems in Using Artifex, A. Khodabandeh, RD13 technical note #57, CERN, 1993, http://rd13doc.cern.ch/public/doc/Note57/Note57.html

Balestra91. The Obelix On-line Monitor and Display, F.Balestra et al. Proceedings of the Conference Computing for High-Energy Physics Conference 1991, Tsukuba, Japan. ISBN 4-946443-09-6,(539-544)..

Bee92. The CPLEAR Data Acqusition and Control System, C.P.Bee et. al., Proceedings of the Internation Conference on Computing in High-Energy Physics '92, Annecy, France 21-25 Sepetmber 1992, ISBN 92-9083-049-2

Berners-Lee87. Experience with Remote Procedure Call in Data Acquisition and Control, T.J.Berners-Lee, 5th Conference on Real-Time Computer Applications in Nuclear Particle and Plasma Physics, San Francisco, 12-14 May 1987.

Birman90. The ISIS project: Real experience with a fault tolerant programming system. K.P.Birman and Robert Cooper, European SIGOPS Workshop, September 1990; also available as Cornell Univ. Computer Science Dept. Tecn. Report TR90-1138.

Birman96. Building secure and reliable network applications, Kenneth P. Birman, Manning,1996, ISBN 1-884777-29-5

Blobel83. Databases and bookkeeping for HEP experiments, ECFA - V. Blobel et al., Rutherford report RL-83-085 (1983).

Brun89. PAW, a general purpose software tool for data analysis and presentation, R. Brun et al, Proceedings of the Conference on Computing in High-Energy Physics, Oxford 1989 (North Holland, 1989).

Bruno95. Model Based Software Engineering, G. Bruno, Chapman & Hall, 1995. ISBN 0 412 48670 9.

Camac92. CAMAC Instrumentation and Interface Standards, IEEE, Inc., 1992.

Caprini96. Java language evaluation for DAQ status display, Mihai Caprini, Zuxuan Qian, 16 September 1996, ATLAS DAQ and Event Filter Prototype "-1" Project Technical Note 10, http://atddoc.cern.ch/Atlas/Notes/010/Note010-1.html

Cattell93. The Object Database Standard, ODBMG-93, Edited by R.G.G. Cattell, ISBN 1-55860-302-6, Morgan Kaufmann.

Dec87. VAX/VMS Introduction to System Services, chapter 12. Digital Equipment Corp. (1987)

Dec88. DECwindows - DEC, VMS DECwindows Technical Summary, order number ZK4727 (DEC 1988).

Dec91. DECWindows Digital Equipment Corporation. VMS DECWindows Motif Guide to Application Programming, August 1991.

Fastbus83. FASTBUS modular high speed data acquisition and control system, DOE/ER - 0189. U.S.Dept. of Energy, Washington, DC (1983); published as IEEE Std. 960.

Ferrato93. Error Message Facility for RD13, Daniele Ferrato, Bob Jones, RD13 Technical Note 6, 16 April 1993, http://rd13doc.cern.ch/public/doc/Note6/Note6.html.

Fresco94. C++ Report Magazine, Fresco Column October 1994.

Fumagalli93. User's Guide to the Artifex DAQ in the RD13 Laboratory, A. Khodabandeh, G. Fumagalli, RD13 technical note #51, CERN, 20-Jan-93, http://rd13doc.cern.ch/public/doc/Note51/Note51.html

Gaspar93. A Distributed Information Management System for DELPHI Experiment at CERN, C.Gaspar, M.Donszelmann, Proceedings of the IEEE Eight Conference REAL TIME `93 on Computer Applications in Nuclear, Particle and Plasma Physics, Vancouver, June 8-11, 1993.

GemStone92. An Introduction to GemStone V3.0. Servio Corporation 1992. 20575 N.W. von Neumann Drive Beaverton, OR 97006, USA. http://www.gemstone.com/

Goodheart91. UNIX Curses Explained. Goodheart, B., Prentice Hall, 1991.

Green89. The ADAMO Data System, M.G. Green, RHBNC 88-01, Royal Holloway and Bedford College, Egham (1989).

Intellicorp93. Kappa CommManager System Guide, Version 3.0 Beta-2, Publication Number: K3.0b2-KMSS-2, August 1993, Intellicorp Inc.1975 El Camino Real West, Mountain View, CA 94040, USA. http://www.intellicorp.com/

Itasca92. ITASCA Distributed Object Database Management System. Technical Summary for Release 2.1. Itasca Systems, Inc. 1992. IBEX Object Systems, SA International Business Park, 4ème blvd, bât Héra, F-74160 Archamps, France http://w3.iprolink.ch/ibexcom/

Jones86. OSP Users' Guide, R.Jones DD/OC CERN Geneva, 17 December 1986.

Jones92. Use of ISIS and Meta, R.Jones et al., Second International Workshop on Software Engineering, Artificial Intelligence and Expert Systems for High Energy and Nuclear Physics (AIHEP'92) conference, La Londe-les-Maures, France in January 1992. ISBN 981-02-1122-8.

Jones93. Testing the interface between Artifex and non-Artifex applications, R Jones, E. Sanchez-Corral, RD13 technical note #62, CERN, March 22, 1993, http://rd13doc.cern.ch/public/doc/Note62/Note62.html

Jones94. Improvements in the Artifex based RD13 Run Control system, R. Jones, E.Sanchez-Corral, RD13 technical note #103, CERN, January 1994, http://rd13doc.cern.ch/public/doc/Note103/Note103.html

Jones95. The RD13 DAQ System and the Object Management Workbench, Bob Jones, Proceedings of the 1995 CERN School of Computing, Arles, France, 20 August - 2 September 1995, CERN 95-05 25 October 1995.

KL94. XRT Builder Guide & Reference Manual, Ref No. BLGDE-GRAPH/M/240-07/94. KL Group. 260 King Street East, Toronto, Ontario, Canada M5A 1K3. http://www.klg.com

Kaplan93. A comparison of queueing, cluster and distributed computing systems, Kaplan, Joseph A. Nelson, Michael L., Oct 01, 1993, NASA Langley Research Center (Hampton, VA, United States), NASA-TM-109025

Khodabandeh93. Performance Study of the Artifex based DFP, A. Khodabandeh, G. Mornacchi, RD13 technical note #50, CERN, 20-Jan-93, http://rd13doc.cern.ch/public/doc/Note50/Note50.html

Kolos96. Inter-component communication in the ATLAS DAQ back-end software (evaluation of the ILU multi-language object interface system), Kolos Sergue, 21 June 1996, ATLAS DAQ and Event Filter Prototype "-1" Project Technical Note 3, http://atddoc.cern.ch/Atlas/Notes/003/Note003-1.html

Le Goff95. CICERO: Control information system concepts based on encapsulated real-time objects. J.M. Le Goff et al. CERN/DRDC/93-50,CERN/LHCC/95-15.

Mapelli90. A Scalable Data Taking System at a Test Beam for LHC, L.Mapelli et al., CERN/DRDC/90-64/P16, CERN/DRDC/90-64/P16 Add.1, CERN/DRDC/90-64/P16 Add.2 (1990).

Mapelli92. RD13 Status Report, L.Mapelli et al., CERN/DRDC/92-13, 13 March 1992.

Mapelli93. RD13 Status Report, L.Mapelli et al., CERN/DRDC 93-25, 5 May 1993.

Mapelli94. RD13 Status Report, L.Mapelli et al., CERN/DRDC 94-24, 9 May 1994.

Mapelli95. RD13 Status Report, L.Mapelli et al., CERN/LHCC 95-4, LCRB Status Report/RD13, 1 August 1995.

Marzullo91. Tools for Constructing Distributed Reactive Systems, Marzullo, K., and M. Wood, Technical Report TR91-1193. Department of Computer Science, Cornell University, February 1991.

Matheys89. Model Process Control User Manual, Jean-Pol Matheys, Version 3.1, CERN DD-OC, 19-07-1989.

Microware96. OS-9 for 68K Technical Overview, white paper, Microware Systems Corporation 1900 NW 114th Street Des Moines, IA 50325, USA. http://www.microware.com (1996).

Microware97. OS9 - Microware Systems Corporation, OS-9/68000 Operating System Technical Manual, order number PMN-OST68. http://www.microware.com (1997).

Mornacchi87. Architecture of the Model Project Human Interface, G. Mornacchi, CERN DD/87/14 (1987).

Mornacchi92. RD13 DataBase FrameWork, G. Mornacchi, RD13 Technical note 11, 1 May 1992, http://rd13doc.cern.ch/public/doc/Note11/Note11.html.

OMG95. Object Management Group. The Common Object Request Broker: Architecture and Specification. OMG, Inc., Jul. 1995. Version 2.0 (Revision 96-08-04).

OMG96. Object Management Group. CORBA services: Common Object Services Specification. OMG, Inc., Mar. 1995. Revised Edition Nov. 1996.

OSF93. Open Software Foundation. OSF/Motif Programmer's Guide, 1.1 edition. ISBN 0-13-643115-1 (1993).

Ontos92. Ontos, Inc. Ontos reference manual. 1992.

PASS94. Petabyte Access Storage Solutions. The PASS Project Architectural Model. Proceedings of CHEP94. Lawrence Berkeley Laboratory LBL-35822; CONF-940492; UC-405.

RD4596. RD45 - A Persistent Object Manager for HEP, The RD45 collaboration, CERN/LHCC 96-15, LCRB Status Report /RD45, 22 February 1996.

Rimmer87. A Database for FASTBUS, E.M. Rimmer, preprint CERN/DD/8723 (1987).

Rimmer90. Databases for Large Detector Systems - Experiences at LEP and Future needs, E.M. Rimmer, CERN CN/90/7, April 1990.

Sendall86. Some notes on the signalling of occurrences, D.M Sendall, MODEL/Design 23, CERN/DD, 27 Feb 1986.

Stratus96. Stratus Computer, Inc.55 Fairbanks Boulevard Marlboro, Massachusetts, U.S.A. (1996). http://www.stratus.com/

Touchard96. Evaluation of the Message Passing Interface Standard for the Atlas Backend Software, F. Touchard, October 7, 1996, ATLAS DAQ and Event Filter Prototype "-1" Project Technical Note 11, http://atddoc.cern.ch/Atlas/Notes/011/Note011-1.html

Unison97. Unison Load Balancer, Mardall House, Grnd Flr, Vaughan Road, Harpenden, Hertfordshire AL5 4HU, United Kingdom http://www.unison.com/index.html (1997).

VMEbus85. VMEbus Specification Manual Rev.C, VMEbus Intl. Trade Assoc., Scotsdale, Ariz. (1985).

Vascotto89. Inter-process control transactions, A. Vascotto, CERN DD-OC-OS, 14 June 1989.

Wenaus89. The L3 Online Window Manager Facility (WM), Torre Wenaus, June 9 1989, L3 Online Note 807.

WenausT89. The L3 Cluster Process Communication Facility (CPC), Torre Wenaus, L3 document note #806 (1989).