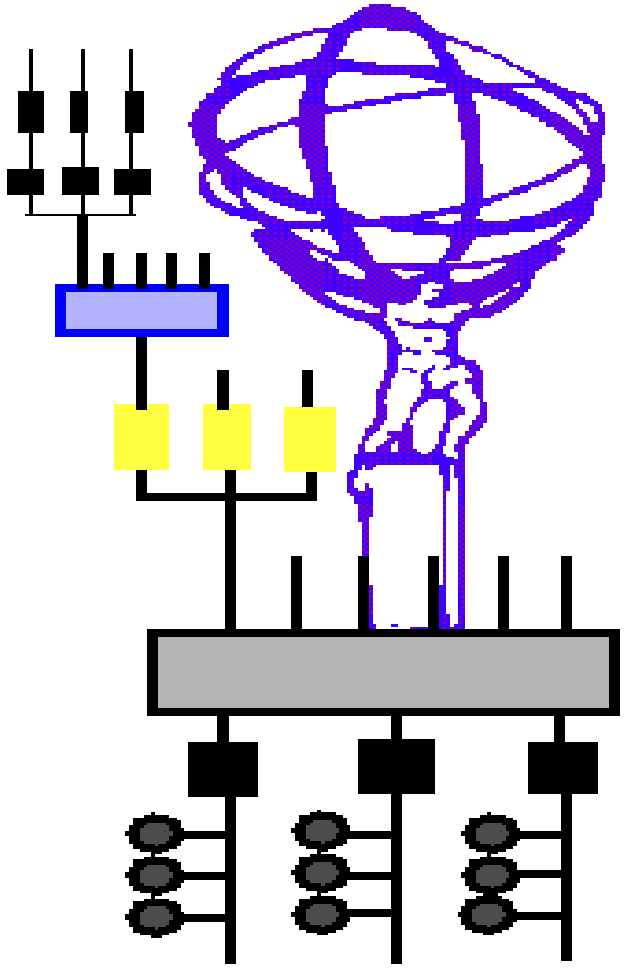
ATLAS

DAQ/EF Prototype -1 Project

Back-end DAQ

Back-End Summary Document

Document Version:	1.2
Document Issue:	Alpha
Document Edition:	English
Document Status:	Draft
Document ID:	ATLAS Internal Note DAQ-No-xxx
Document Date:	16 December 1999



European Laboratory for Particle Physics
Laboratoire Européen pour la Physique des Particules
CH-1211 Genève 23 - Suisse

This document has been prepared using the Software Documentation Layout Templates that have been prepared by the IPT Group (Information, Process and Technology), IT Division, CERN (The European Laboratory for Particle Physics). For more information, go to http://framemaker.cern.ch/.

Abstract

This document provides a summary of the ATLAS DAQ Back-end software sub-system which is part of the DAQ/Event Filter Prototype "-1" project for the ATLAS experiment. The back-end software encompasses all the software to do with configuring, controlling and monitoring the DAQ but specifically excludes the management, processing or transportation of physics data. The software described in this document is the work of the ATLAS back-end DAQ sub-group including the following people:

CERN: D.Burckhart, M.Caprini (on leave from Institute of Atomic Physics, Bucharest), R.Jones, S.Kolos (on leave from PNPI, St Petersburg), L.Mapelli, D. Schweiger, I.Soloviev (on leave from PNPI, St Petersburg)

CPPM, IN2P3-CNRS, Marseille: L.Cohen, P.Y.Duval, D. Laugier, R.Nacasch, Z.Qian

Institute of Atomic Physics, Bucharest: E.Badescu

JINR, Dubna: I.Alexandrov, V.Kotov, V. Roumiantsev

University of Geneva, Geneva: L.Moneta

LIP, Lisbon: V. Amaral, A.Amorim, C. Ribeiro

NIKHEF, Amsterdam: R.Hart

PNPI, Gatchina, St. Petersburg: A. Kazarov, Y.Ryabov

Document Control Sheet

Table 1 Document Control Sheet

Document	**Title:**	DAQ/EF Prototype -1 Project Back-End Summary Document
	Version:	1.2
	Issue:	Alpha
	Edition:	English
	ID:	ATLAS Internal Note DAQ-No-xxx
	Status:	Draft
	Created:	11 October 1999
	Date:	16 December 1999

Table 1 Document Control Sheet

	Keywords:	Back-end DAQ Prototype -1 Software ATLAS		
Tools	**DTP System:**	Adobe FrameMaker	**Version:**	5.5
	Layout Template:	Software Documentation Layout Templates	**Version:**	V2.0 - 5 July 1999
	Content Template:	--	**Version:**	--
Authorship	**Coordinator:**	Bob Jones		
	Written by:	ATLAS Back-end DAQ group		

Table of Contents

Chapter 1
Introduction

This document provides a summary of the back-end DAQ software.

1.1 Overview

This document provides a summary of the ATLAS DAQ Back-end software sub-system which is part of the DAQ/Event Filter Prototype "-1" project [2] for the ATLAS experiment. The back-end software encompasses all the software to do with configuring, controlling and monitoring the DAQ but specifically excludes the management, processing or transportation of physics data. Figure 1 is a basic context diagram for the back-end software showing the exchanges of information with the other sub-systems. This context diagram is very general and some of the connections to the other sub-systems have not been implemented in the prototype DAQ/Event Filter Prototype "-1-" project.

The Back-end software described in this document is implemented according to a design [4] intended to satisfy the requirements defined in the User Requirements Document [3]. The intended audience are project reviewers and developers of the back-end software. It is the product of a ATLAS DAQ Back-end software group and is drawn from the more detailed individual component documents.

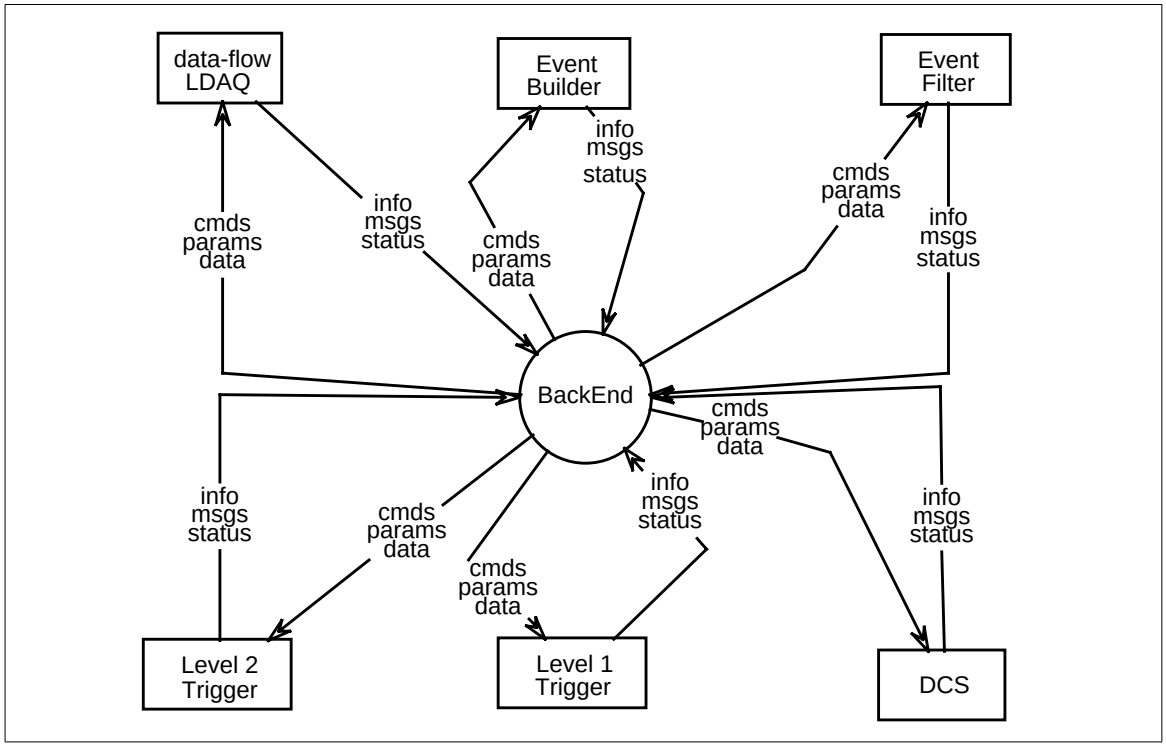


Figure 1 Back-End DAQ software context diagram showing exchanges with other sub-systems

Figure 2 shows on which processors the back-end software is expected to run. Note that the back-end software is not the only software that will be run on such processors. A set of operator workstations, situated after the event filter processor farm, dedicated to providing the man-machine interface and hosting many of the control functions for the DAQ system will also run back-end software. The back-end software is essentially the “glue” that holds the various sub-systems together. It does not contain any elements that are detector specific as it is to be used by all possible configurations of the DAQ and detector instrumentation.

The back-end software is but one sub-system of the whole DAQ system and it must co-exist and co-operate with the other sub-systems.

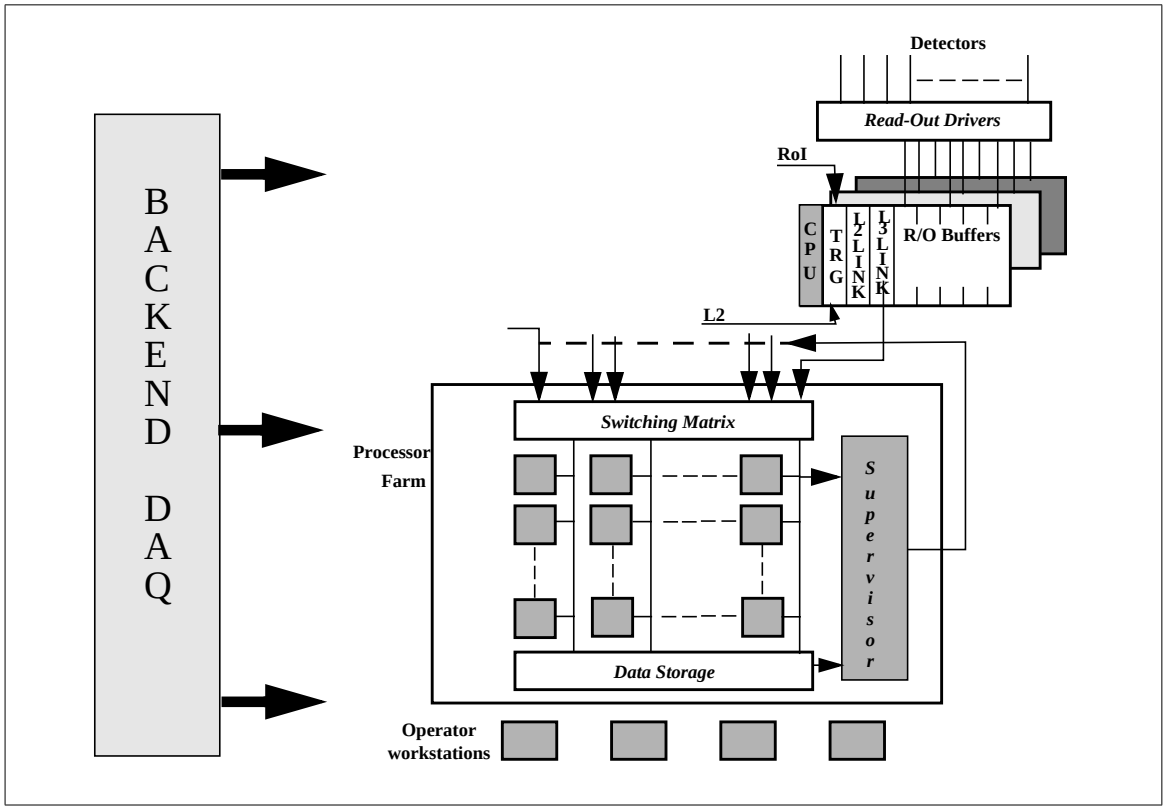


Figure 2 ATLAS DAQ/Event Filter Prototype “-1” architecture

1.2 Operational environment

It is expected that this environment will be a heterogeneous collection of workstations, PCs running Windows NT or Linux and embedded systems (known as LDAQ processors) running various flavours of real-time UNIX operating systems (e.g. LynxOS) connected via a Local Area Network (LAN).

A great number of hardware components have to be used to provide the necessary computing power. These will be back-end and LDAQ processors loosely coupled by local networks and data flow channels. We assume that network connections (e.g. ethernet or replacements) running the most popular protocols (e.g. tcp / ip) are available to all the target platforms for exchanging control information and that its use will not interfere with the physics data transportation performance of the DAQ.

The ATLAS prototype DAQ system will need to be able to run using all or only a part of its sub-systems. It will be assembled in a step by step manner, according to financial and technical dependencies and constraints. A high degree of modularity is needed to connect, disconnect or add new components at will.

Many groups of people will interact at the various hardware and software levels, and so we have to foresee a significant level of sub-system unavailability and that this shall be detected and tolerated during DAQ system startup. Hence checking procedures shall be a concern to detect such configuration problems. The failure of an individual component should not affect the operation of other components.

Chapter 2
Components

2.1 The software component model

The user requirements gathered for the back-end sub-system have been divided into groups related to activities providing similar functionality. The groups have been further developed into components of the back-end with a well defined purpose and boundaries. The components have interfaces with components and external systems, specific functionality and their own architecture.

From analysis of the components, it was shown that several domains recur across all the components including data storage, inter-object communication and graphical user interfaces.

2.2 Core components

The following components are considered to be the core of the back-end sub-system. The core components constitute the essential functionality of the back-end sub-system and have been given priority in terms of time-scale for development in order to have a baseline sub-system that can be used for integration tests with the data-flow sub-system and event filter.

2.2.1 Run control

The run control component controls the data taking activities by coordinating the operations of the DAQ sub-systems, back-end software components and external systems. It has user interfaces for the shift operators to control and supervise the data taking session and software interfaces with the DAQ sub-systems and other back-end software components. Through these interfaces the run control can exchange commands, status and information used to control the DAQ activities.

2.2.2 Configuration database

A data acquisition system needs a large number of parameters to describe its system architecture, hardware and software components, running modes and the system running status. One of the major design issues of Atlas DAQ is to be as flexible as possible, parameterized by the contents of databases.

2.2.3 Message reporting system

The aim of the Message Reporting System (MRS) is to provide a facility which allows all software components in the ATLAS DAQ system and related sub-systems to report error messages to other components of the distributed DAQ system. The MRS performs the transport, filtering and routing of messages. It provides a facility for users to define unique error messages which will be used in the application programs.

2.2.4 Process manager

The purpose of the process manager is to perform basic job control of software components of the DAQ. It is capable of starting, stopping and monitoring the basic status (e.g. running or exited) of software components on the DAQ workstations and LDAQ processors independent of the underlying operating system. In this component the terms process and job are considered equivalent.

2.2.5 Information service

The Information Service (IS) provides an information exchange facility for software components of the DAQ. Information (defined by the supplier) from many sources can be categorised and made available to requesting applications asynchronously or on demand.

2.3 Trigger / DAQ and detector integration components

Given that the core components described above exist, the following components are required to integrate the back-end with other on-line sub-systems and detectors.

2.3.1 Partition and resource manager

The DAQ contains many resources (both hardware and software) which cannot be shared and so their usage must be controlled to avoid conflicts. The purpose of the Resource Manager is to formalise the allocation of DAQ resources and allow groups to work in parallel without interference. The Partition manager is intended to extend this functionality to whole partitions.

2.3.2 Integrated Graphical User Interface

The integrated graphical user interface (IGUI) allows the operator to control and monitor the status of the current data taking run in terms of its main run parameters, detector configuration, trigger rate, buffer occupancy and state of the sub-systems.

2.3.3 Online bookkeeper

The purpose of the online bookkeeper is to archive information about the data recorded to permanent storage by the DAQ system. It records information on a per-run basis and provides a number of interfaces for retrieving and updating the information.

2.3.4 Event dump

The event dump is a monitoring program with a graphical user interface that samples events from the data-flow and presents them to the user in order to verify event integrity and structure.

2.3.5 Test manager

The purpose of the test manager is to organise individual tests for hardware and software components. The individual tests themselves are not the responsibility of the test manager which simply assures their execution and verifies their output. The individual tests are intended to verify the functionality of a given component. They will not be used to modify the state of a component or to retrieve status information. Tests are not optimized for speed or use of resources and are not a suitable basis for other components such as monitoring or status display.

2.3.6 Diagnostics package

The diagnostics package uses a knowledge base [22] and the tests held in the test manager to diagnose problems with the DAQ and verify its functioning status. By grouping tests into logical sequences, the diagnostic framework can examine any single component of the system (hardware or software) at different levels of detail in order to determine as accurately as possible the functional state of components or the entire system. In case any faults are found, it reports the diagnosis and suggests actions necessary to restore the affected component.

Chapter 3 Performance and Scalability

This chapter describes the measured performance and scalability for the back-end components and the overall back-end software suite.

3.1 Component Unit Tests

For all core components an implementation exists and functionality and performance tests have been performed. For most detector integration components an implementation is available and testing has started and is expected to be completed soon. Each component is subjected to unit tests to assess its functionality, performance, scalability and reliability. For each component a test plan has been prepared and the test results have been reported [6], [10], [13], [14]. The integration of the core components was made in a step-wise manner, according to the dependencies between components and the underlying external packages (both commercial and shareware) see Appendix B.1. Integration and scalability tests are based on the two most relevant scenarios for an integrated DAQ system representing the likely configurations for the DAQ/EF Prototype - 1 project and the final ATLAS DAQ/EF system.

For further information on the results of the unit tests for back-end components refer to [24].

3.2 Integration Tests

The integration tests bring together all the core components and several Trigger/DAQ/Detector integration components to simulate the control and configuration of data taking sessions. The steps involved in performing integration tests are:

- Complete the content of the configuration database to define all the necessary software elements.
- Execute a script (called play_daq) which is responsible for starting and shutting down the DAQ. The script can operate in two modes:

operator:
- script initialises back-end and then displays IGUI to allow the operator to exercise the partition. On exiting the IGUI the script closes down all back-end services.

automatic:

- same as normal mode but rather than allowing the operator to exercise the partition, commands are send directly from the script to take the partition to the Running state, perform some run cycles then shut it down. The script keeps track of the time needed to execute the various phases of the operation and outputs them when it exits. The values represent the basic measured points for the back-end scalability and performance tests.

3.2.1 play_daq script

The detailed steps performed by the script in automatic mode are shown in Figure 3 and described below:

- Start the back-end server processes (e.g. MRS server, IPC server etc.) required for any partition.
- Launch configuration specific processes using the DAQ supervisor (via the process manager).
- Marshal the hierarchy of run control controllers from their *Initial* to *Running* state by sending commands to the Root controller.
- The steps upto this point represent the actions required to star a data-taking session and bring the DAQ into the *Running* state.
- Marshal the hierarchy of run control controllers back to the *Configured* state. This represents stopping a data taking run.
- Marshal the hierarchy of run control controllers back to the *Running* state. This represents re-starting a data taking run for the same configuration.
- Marshal the hierarchy of run control controllers back to the *Initial* state.
- Stop the configuration specific processes using the DAQ supervisor (via the process manager).
- Stop the back-end server processes.

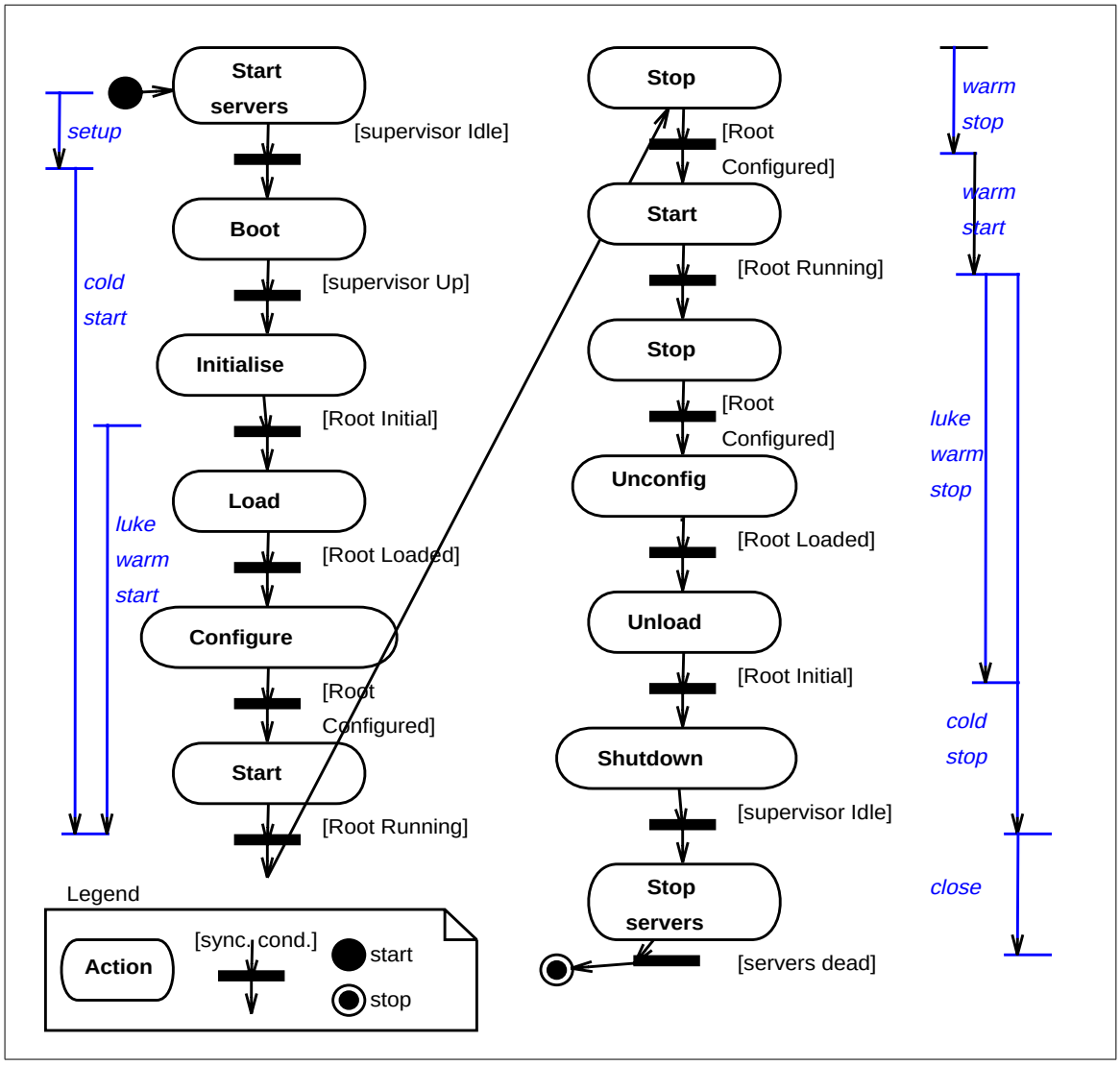


Figure 3 UML Activity diagram showing the actions performed by the play_daq script

3.2.2 Timing measurements

Set-up start back-end servers

When all machines are booted, measure the time it takes to start the back-end servers (MRS, Information Service etc.). This test corresponds to the actions needed at the start of a data taking session.

Close stop back-end

Measure time taken to do the reverse of the Set-up test. This test corresponds to the actions needed at the end of a data taking session.

Cold start start a configuration

When all back-end servers and a Process Manager agent is running on each machine, measure the time it takes to start the configuration specific processes including all controllers with their associated LDAQ emulators, then do all necessary transitions to get to the *Running* state. This test corresponds to the actions needed to start of a configuration.

Cold stop stop a configuration

Measure time taken to do the reverse of the cold start test. This test corresponds to the actions needed stop a configuration.

Lukewarm start restart a configuration

When all necessary processes are started and the controllers are in the *Initial* state, measure the time taken to get to the *Running* state. This test corresponds to the actions needed when re-starting a configuration, such as after a *Reset* command has been given to the run control.

Lukewarm stop reset a configuration

Measure time taken to do the reverse of the lukewarm start test. This test correspond to the actions needed when stopping a configuration.

Warm start start a single run

When all processes are alive and all controllers in the *Configured* state, measure the time taken to get to the *Running* state. This test corresponds to the actions needed when starting a new run with the same configuration.

Warm stop stop a single run

Measure time taken to do the reverse of the warm start test. This test corresponds to the actions needed to stop a run.

3.2.3 Back-end servers

The set of back-end servers started by the script for each configuration on the operator works tat ion includes:

- global IPC server
- partition IPC server
- RM server (resource manager)
- RDB server (remote database access)
- MRS server (Message Reporting System server)
- RC IS server (Information Service server for run control)
- PMG IS server (Information Service server for process manager)
- DF IS server (Information Service server for data-flow statistics)
- IGUI (integrated Graphical user Interface for the operator)
- MRS receiver (Message Reporting System receiver: writes all messages to a log file)
- DAQ Supervisor

- PMG agent (daemon for creating and stopping processes on a machine)

3.3 Overview of Scenarios Tested

The procedure described above has been followed for all the configuration show in Table 2. Configurations named NxN (i.e. 2x2, 6x6) represent the data-flow organisation of the DAQ VME crates, i.e. Read-Out Crates x SubFarm Crates + Event Builder. Details of the characteristics of the configurations are given below.

3.3.1 run control

Configurations were run with the controllers in various locations:

centralised all the controllers were run on the operator works tat ion (Figure 4)

distributed the controllers associated with ldaq emulators were run on the same machine as the ldaq emulators (Figure 5)

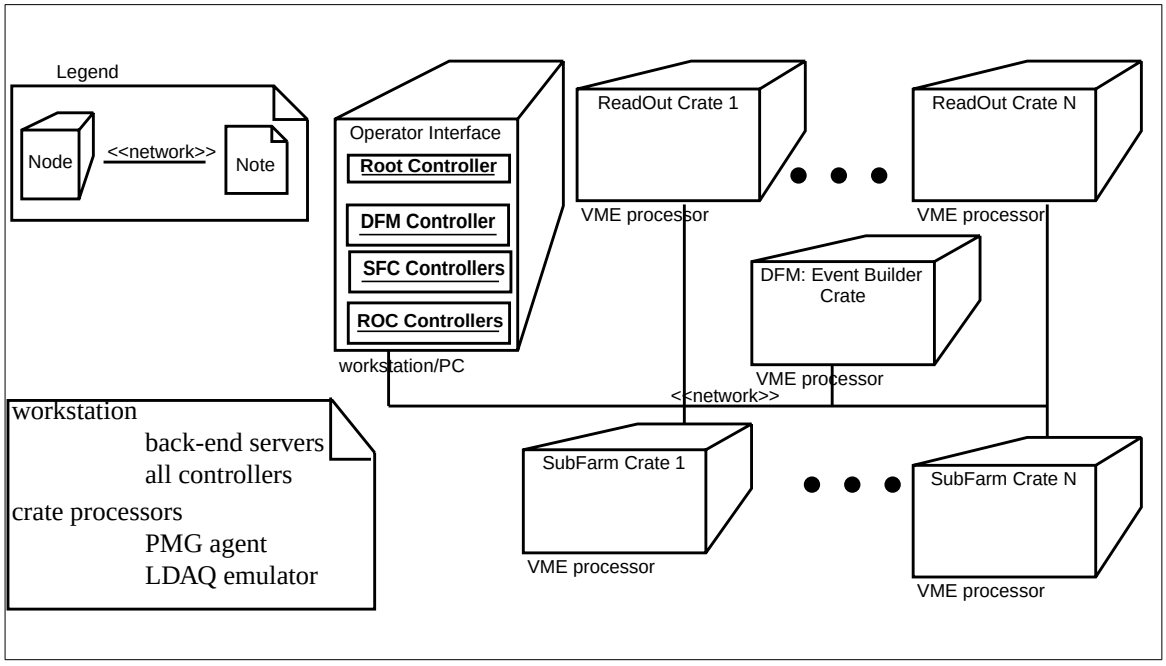


Figure 4 UML deployment diagram showing centralised controller organisation for 2x2 configuration

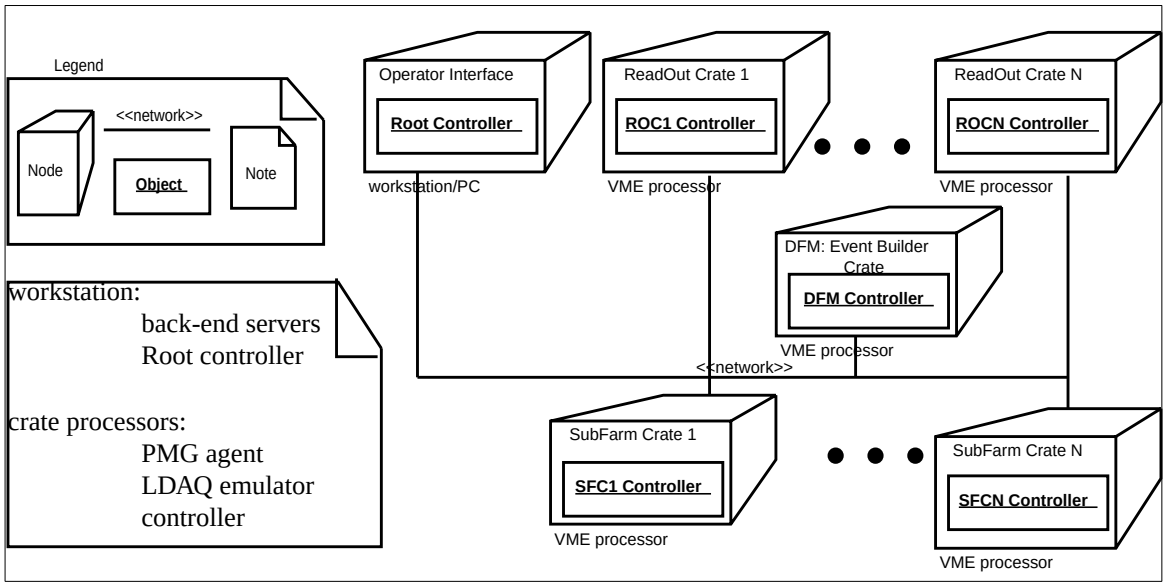


Figure 5 UML deployment diagram showing distributed controller organisation for 2x2 configuration

The number of controllers increases with the size of the configuration tested. All configurations used a 2-tier hierarchy with a single Root controller controlling a set of sub-controllers. For example, the controller hierarchy for the single host and 2x2 configurations is shown in Figure 6.

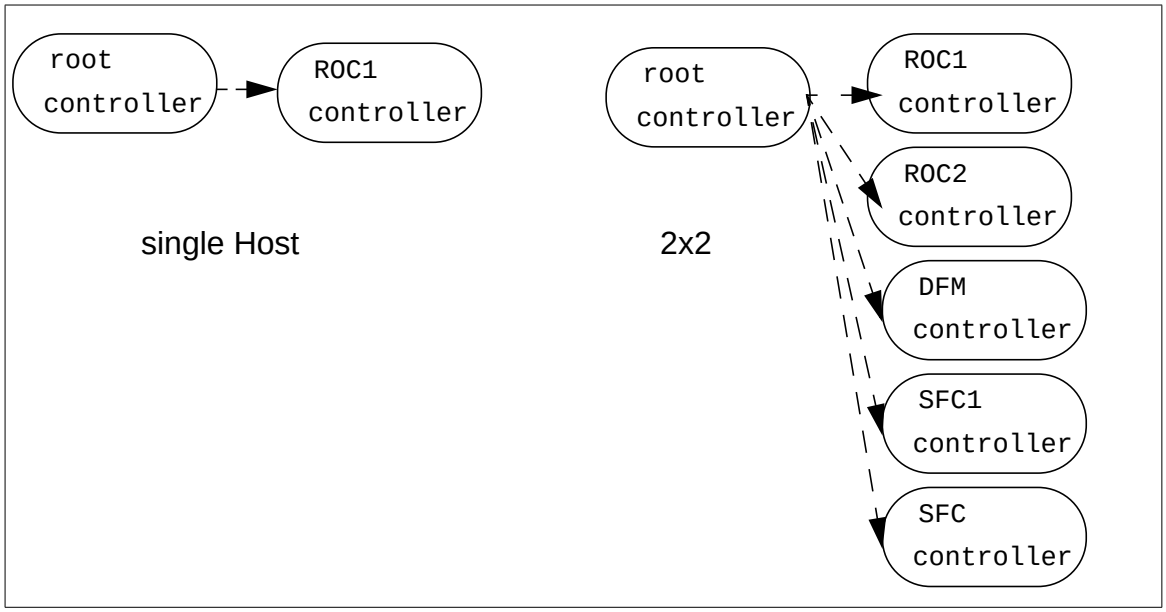


Figure 6 Controller hierarchies for single host and 2x2 configurations

3.3.2 LDAQ emulator

To simulate the interface with the data-flow part of the DAQ, an LDAQ emulator has been developed. The LDAQ emulator allows testing of the back-end software sub-system independently of the data-flow software. The LDAQ emulator:

- exchanges run control messages with an associated controller via a tcp / ip link [25],
- sends MRS messages,
- sends IS information to simulate data-flow statistics information about crate modules,
- performs no VME access,
- contains no data-flow software.

3.3.3 Process Management

Two forms of process management were used for the various configurations:

LynxOS based configurations the PMG agents were started by the script

The script calls an expect script to make a remote login into the ldaq processor and start the agent. The list of machines on which to start the agents is retrieved from the configuration database.

Linux based configurations the PMG agents were started by hand

The agents were started from an X terminal logged into the remote machine during the set-up phase of the script

During test-beam or final experiment usage, it is envisaged to start the PMG agents during the boot sequence of the processors. This is not possible during development since many agents are started for testing purposes. In this case, the set-up phase of the script will be much quicker because the starting of the agents represents the largest fraction of the elapsed time.

Table 2 Back-end integrated configuration scenarios

Configuration	number of controllers	platform and releases
Single Host	2: Root + ROC1[a]	LynxOS[b] (+ 1 Solaris work-station) back-end release 0.0.6
2x2 centralised	6: Root, DFM[c], 3 ROCs, 3 SFCs[d]	
2x2 distributed	6	
6x6 centralised	14: Root, DFM, 6 ROCs, 6 SFCs	
6x6 distributed	14	
10x10 centralised	22: Root, DFM, 10 ROCs, SFCs	

Table 2 Back-end integrated configuration scenarios

Configuration	number of controllers	platform and releases
1 PC	2: Root + 1 crate	Linux[e] back-end release 0.0.7
2 PCs	3: Root + 2 crates	
3 PCs	4: Root + 3 crates	
4 PCs	5: Root + 4 crates	
5 PCs	6: Root + 5 crates	
6 PCs	7: Root + 6 crates	
7 PCs	8: Root + 7 crates	
8 PCs	9: Root + 8 crates	

a. Read Out Crate

b. LynxOS 2.5.1 running on RIOs VME single board computers (PPC 100MHz 32MB and PPC 200MHz 64MB)

c. Data Flow Manager (i.e. event builder)

d. SubFarm Crate

e. RedHat 6.0 running on PCs (Intel Pentium III 450MHz 128MB)

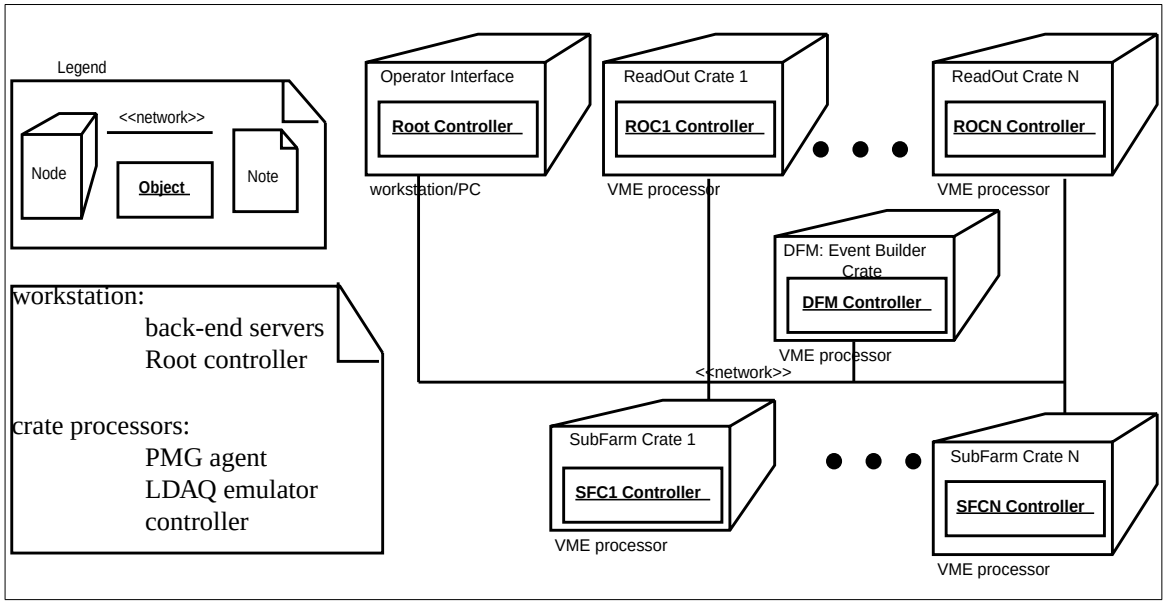


Figure 7 UML deployment diagram showing distributed controller organisation for 2x2 configuration

3.4 Test results

The results from running the tests described in Section 3.2 on the configurations described in Section 3.3 are presented in Figure 8 and Figure 9.

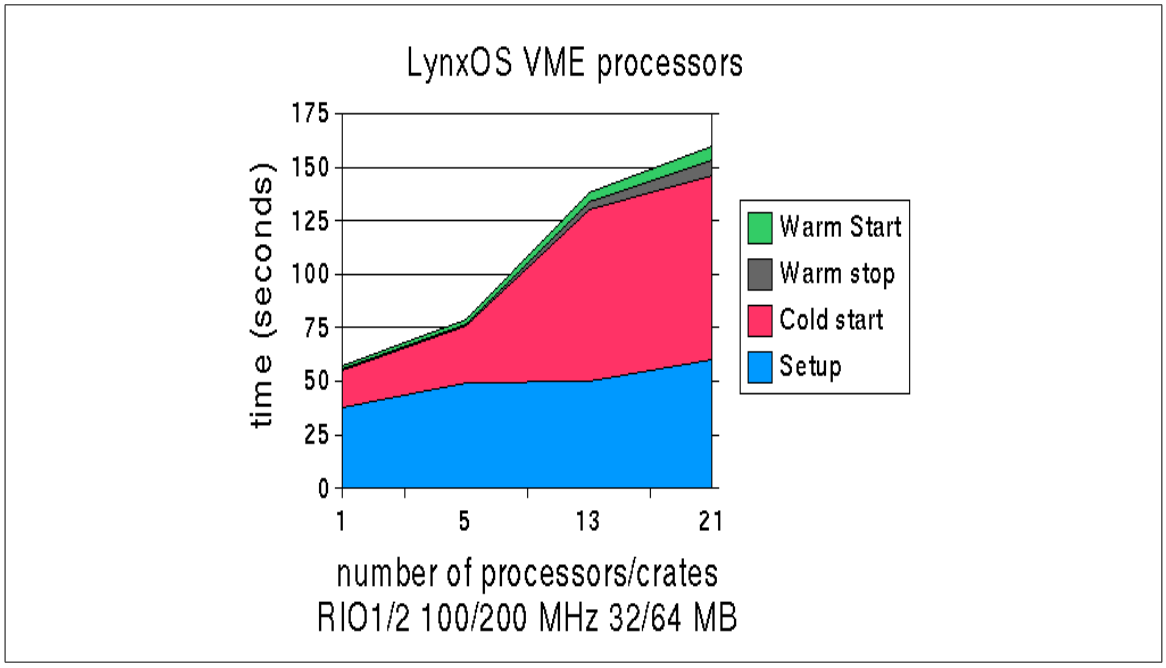


Figure 8 Integrated test results for LynxOS based configurations

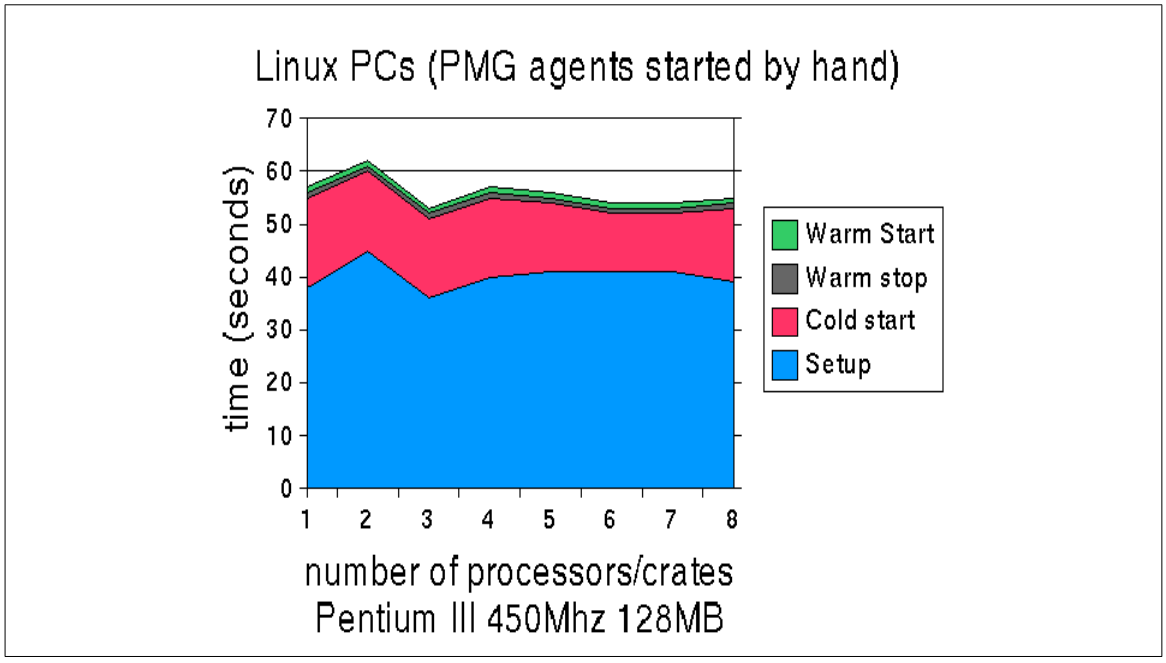


Figure 9 Integrated test results for LynxOS based configurations

3.5 Discussion of the Tests

The tests described above represent a first attempt at quantifying the performance and scalability of the integrated back-end software using an official release of the software [27].

From the results of the integrated tests on various configurations, the following insights can be obtained:

- The machines used for the tests were not dedicated to the task and were being used by other developers at the same time to run other jobs.
- The time taken to start and stop processes is dependent on the operating system, machine specification and system load. The times recorded were far shorter on Linux PCs that LynxOS VME processors.
- Once all the processes have been started, the time taken to exercise the finite state machines of the controllers remains constant. This indicates that the mechanism used for the distributed control scales well with the size of the configuration used.
- The use of the Information Service, Message Reporting System and Configuration Database has a negligible effect on the performance of the DAQ.
- The means used to start the Process Manager agents remotely from the test script is a temporary solution. Currently an expect script is called for each agent that logs into the target host and starts the agent process. This is necessary during development because multiple agents are being started on each machine for testing purposes. In test-beam and production use, only one agent will be started on each machine and its initialization could be put in the boot sequence of the host. The result of the current situation is that the times recorded for set-up action depict a worse-case scenario.
- The times recorded for stopping processes (i.e. luke warm stop and cold stop) are sometimes longer than the times needed to start processes. This result is unexpected and requires further investigation.
- The test results in Figure 9 show, that even for the largest configuration tested the execution time of the initialisation, operation and shutdown is in an acceptable range (maximum 1 minute for a Linux based 8 processor system).
- For the true setup (i.e including data-flow software and hardware) the communication time with the controlled components and their execution time will be in addition to this value.

Chapter 4
Summary

4.1 Integration Tests

Following individual component unit tests, integrated tests have been performed employing the majority of the components developed in the back-end DAQ. Such integration tests have been performed with the goal of verifying the correct inter-operation of the components, the ability to operate in a distributed, heterogeneous multi-platform environment and gather performance measurements relevant to the operation of the DAQ in a production environment. The test configurations have included workstations, PCs and VME based processors. The operating systems involved were Solaris, LynxOS and Linux. Configurations using up to 21 processors have been tested.

Further testing is required to:

- gather more statistics on the time measurements for the various configurations;
- determine more precisely the reasons for the inferior performance of LynxOS compared to Linux;
- determine the reasons why stopping processes sometimes takes longer than starting processes;
- perform tests on larger configurations if access to suitable hardware is possible.

Future work to further improve the performance of the back-end will concentrate around the play_daq script, daq supervisor and start-up of pmg agents to make more use of available synchronization techniques and not rely on delay timeouts.

An obvious improvement would be to combine the LDAQ and associated controller into a single process. This would reduce the number of processes that need to be started, reduce the number of network messages required to pass commands, simplify synchronization and reduce the amount of software employed.

4.2 Future Work

Development will continue on the individual components of the back-end DAQ software, notably in the area of the Trigger / DAQ and Detector integration components. The developments will be included in the iterative releases of the software which have been produced on a monthly basis and distributed on CD-ROMs.

In the near future, we anticipate closer interaction and integration with other sub-systems of the ATLAS Trigger / DAQ / DCS project. The status of integration to date can be summarised as follows:

Level-1 Trigger Members of the Level-1 community have used release 0.0.5 of the back-end software distributed on CD-ROM to evaluate it applicability in their domain. They have produced very useful feedback which has been used to influence the development of subsequent releases.

Level-2 Trigger Some elements of the Reference Software are based on the back-end core component designs. This will simplify future integration and discussions have been held on how they can make use of the core components though more work is required to understand the use of databases.

Event Filter Integration with MRS, IS and the run control components has been made with Marseille prototype system but more work is required on databases. The approach to be taken in the future will depend on how the 3 event filter prototype systems will develop and the interaction with the common software set that is being produced.

DCS A document describing the interaction between the DCS SCADA commercial software package and the back-end components has been written and published [26]. The approach described in the document needs to be validated via actual implementation and usage.

The strategy for integration suggested for the future is similar to that already employed with the Level-1 Trigger group. Releases of the back-end software will be distributed on CD-ROM to representatives from each sub-system to evaluate the software at their own institutes, at their own pace on their own machines. Their feedback should be channelled through a sub-system contact person. This feedback will be used to influence the development of future releases of the software to accommodate common sub-system functionality.

4.3 Back-end Software Summary

The integration tests described in the previous chapter are intended to provide a version of the back-end system for use within the DAQ / EF Prototype -1 project. The results on performances and scalability are in accordance with the DAQ / EF prototype requirements. Work will continue to cover a wider spectrum of configurations to produce information about eventual limits and boundaries.

The software component model has helped to sub-divide the project into more manageable development tasks and encouraged the study of interactions between different elements of the system. The use of common software technologies [16] for data persistence,

communication etc. has reduced the total programming effort, made the developers aware of the structure of all the components and allowed them to share software.

It is expected that integration with other sub-systems will lead to the identification of possible improvements which, coupled to evolving underlying technologies, will form the basis of further developments of the back-end components. Currently, the ILU CORBA based communication package is used in the project [17] but alternative implementations have recently become available and layered services have also been defined. Some activities to investigate such packages have already started and will continue. We would like to extend the DAQ supervisors capacity for decision making and believe expert systems to be a good candidate technology for implementing the logic of such an intelligent supervisor.

Use of the ATLAS prototype DAQ/EF project with prototype detectors in a test-beam environment will provide the opportunity to determine if the back-end sub-system requirements are relevant, its architecture suitable and the adopted software standards, tools and techniques applicable. Given the longevity of the ATLAS experiment, emphasis has been put on analysis and design of the sub-system since it is these aspects (rather than the actual code itself) which will remain relevant up to and beyond the experiments start-up (2005).

Appendix A
Documentation

Table A.1 shows the document produced at each phase of the software process for every back-end component. Technical Notes are available on the web at http://atddoc.cern.ch/Atlas/Notes/Welcome.html

Table A.1 Back-end published technical notes

Component	Require.	Evaluat.	Design	Implement	Test Plan	Test Report
run control	URD[a],134	7,12,13,24	29	105,107	94	113
config. databases	URD	9	30,54	33,122,135	99	114
Msg. Report. System	URD		32	59	96	121
Information Service	31		31	37	95	118
Process Manager	URD	8	28	81	100	
Inter-Process Comm.	URD	3,11		75	103	123
Resource Manager	52		52	130	131	137
Integrated GUI	URD	4,10				
Online Book-keeper	URD		49	117		
Test Manager	URD		66	111,112		
Diagnostics	106	108	106			
Event Dump	URD					

a. General Back-end User Requirements Document [3]

Appendix B
Software Releases

This appendix documents the release structure and history of the back-end DAQ software.

B.1 Packages

The back-end DAQ software is held in a CVS repository organised as a number of packages and managed by the ATLAS configuration management system, called SRT [15]. Below is the list of packages held in CVS:

ace operating system independent interface to basic services (process creation, tcp / ip sockets, threads, etc.) (external) [21]

be_integ_test back-end component integration tests

chsm Harel state chart code generator (external) [20]

clips expert system (external) [22]

cmdl command line parameter handler (external)[23]

confdb configuration database access libraries and utilities

dvs verification module of diagnostics component

dsa supervision module of diagnostics component

hello example package showing how to use SRT

igui integrated graphical user interface for DAQ operator

ilu Corba implementation (external) [19]

ipc high-level interface to ilu and partitioning

is information service

mrs message reporting system

obk online book-keeper

oks persistent object manager used as the basis for the configuration databases

pmg process manager

rc run control

rdb remote database access via Corba

rm resource manager

tmgr test manager

tools_h++ general purpose C++ library(similar to STL) (external) [18]

xmext A collection of freeware Motif widgets from various sources (external)

B.2 Package dependencies

Figure B.1 shows a simplified view the dependencies between the back-end packages.

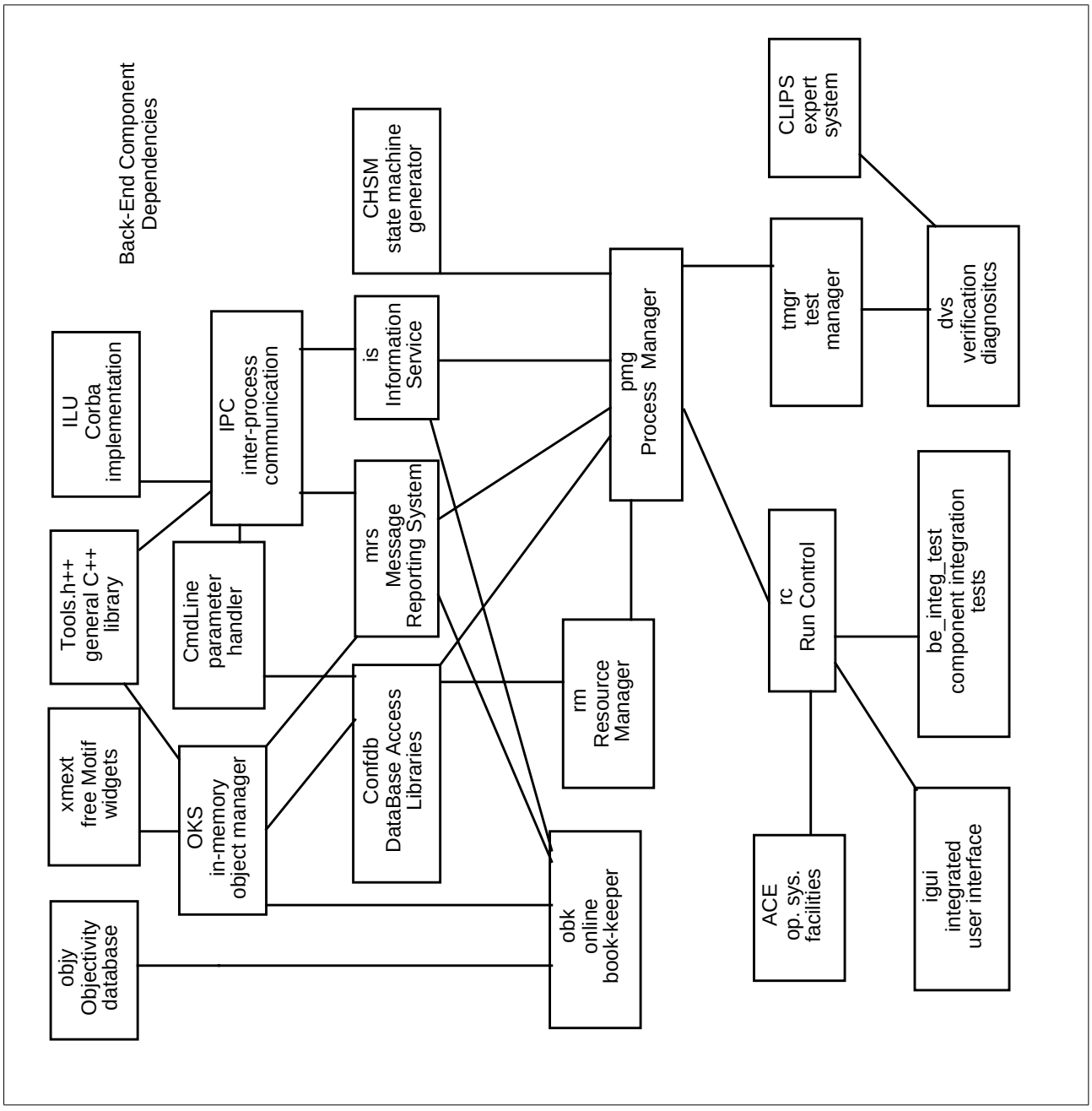


Figure B.1 Back-end DAQ software package depencies

B.3 Releases

Table B.1 shows the platforms supported for each of the public releases.

Table B.1 Back-end public release platforms

Platform	00-00-00 Sep'98	0.0.0 Jan'99	0.0.1 Feb'99	0.0.2 Mar'99	0.0.3 Apr'99	0.0.4 Jun'99	0.0.5 Aug'99	0.0.6 Oct'99	0.0.7 Dec'99
hpux10.20 / g++ 2.7.2	✶	✶							
winnt4.0 / msvc++-5.0	✶	✶							
lynxos2.4.0 / g++2.6-95q2	✶								
lynxos2.5.1 / g++2.6-97q1	✶	✶	✶	✶	✶	✶	✶	✶	✶
solaris2.5.1 / g++2.7.2	✶	✶	✶	✶	✶	✶	✶	✶	✶
solaris2.6 / SunPro-4.2			✶	✶	✶	✶	✶	✶	✶
linux RedHat 5.1 / egcs 1.0		✶	✶	✶	✶				
linux RedHat 6.0 / egcs 1.0						✶	✶	✶	✶

Table B.2 shows which packages are included in each of the public releases

Table B.2 Components included in Back-end public releases

Package	00-00-00	0.0.0	0.0.1	0.0.2	0.0.3	0.0.4	0.0.5	0.0.6	0.0.7
ace	✶	✶	✶	✶	✶	✶	✶	✶	✶
be_integ_test					✶	✶	✶	✶	✶
chsm	✶	✶	✶	✶	✶	✶	✶	✶	✶
clips					✶	✶	✶	✶	✶
cmdl	✶	✶	✶	✶	✶	✶	✶	✶	✶
confdb	✶	✶	✶	✶	✶	✶	✶	✶	✶
dsa									✶
dvs								✶	✶
hello	✶	✶	✶	✶	✶	✶	✶	✶	✶
igui							✶	✶	✶
ilu	✶	✶	✶	✶	✶	✶	✶	✶	✶

Table B.2 Components included in Back-end public releases

Package	00-00-00	0.0.0	0.0.1	0.0.2	0.0.3	0.0.4	0.0.5	0.0.6	0.0.7
ipc	✶	✶	✶	✶	✶	✶	✶	✶	✶
is	✶	✶	✶	✶	✶	✶	✶	✶	✶
mrs	✶	✶	✶	✶	✶	✶	✶	✶	✶
obk				✶	✶	✶	✶	✶	✶
oks	✶	✶	✶	✶	✶	✶	✶	✶	✶
pmg	✶	✶	✶	✶	✶	✶	✶	✶	✶
rc	✶	✶	✶	✶	✶	✶	✶	✶	✶
rdb	✶	✶	✶	✶	✶	✶	✶	✶	✶
rm		✶	✶	✶	✶	✶	✶	✶	✶
tmgr		✶	✶	✶	✶	✶	✶	✶	✶
tools_h++	✶	✶	✶	✶	✶	✶	✶	✶	✶
xmext		✶	✶	✶	✶	✶	✶	✶	✶

Appendix C

Software Process

This appendix describes the software process employed for the development of the back-end DAQ software.

The development has been divided into a number of sequential phases intended to help pace and organise the work. Each phase has been defined to produce an obvious deliverable (i.e. document and / or code) which is reviewed before progressing to the next phase. The phases are: collect requirements; identify and evaluate candidate technologies and techniques capable of addressing the common issues identified from the requirements; produce a design for each component covering the most important aspects; refine the design to add more detail; implement and unit test according to the design; integrate with other components. Figure C.1 shows the phases of this software process and the artifacts produced by each phase including documents used during development, code and user documentation. The deliverables from each phase have been reviewed. initially, reviews took the form of presentations followed by discussions during open meeting with all developers involved in the project. Later more formal inspections were introduced using a system of peer review supported by guidelines and checklists for documentation and code.

The people involved in the back-end DAQ come from many institutes and have not, in general, been able to work full-time on the project. Faced with this situation, we have tried to organise the work along the component structure. Typically, a single institute has taken responsibility for developing a component there by simplifying communication and reducing travel. Such component groups are small (up to a maximum of 5 individuals). The same individuals have tended to follow a single component through the various phases and hence ensured the continuity of the work.

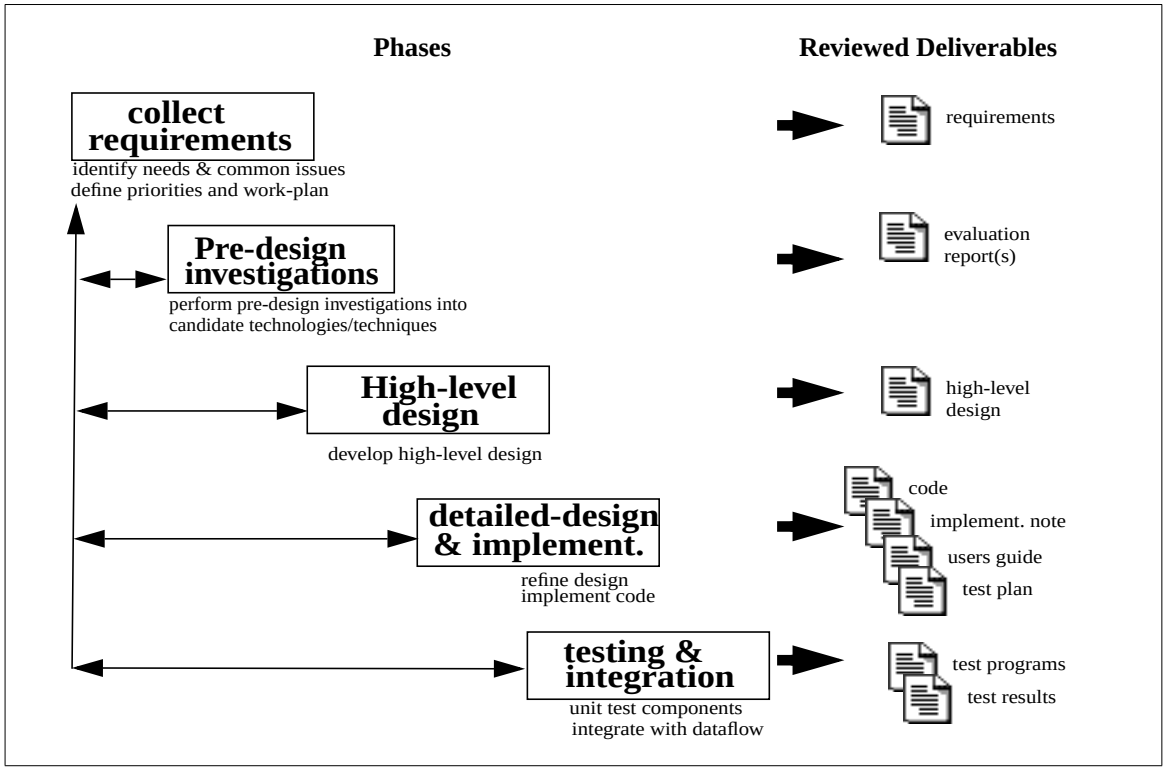


Figure C.1 Software process employed for the development of the Back-End DAQ

Appendix D

References

1 ATLAS Technical Proposal, CERN/LHCC/94-43 (ISBN 92-9083-067-0).

2 G. Ambrosini et al., The ATLAS DAQ and Event Filter prototype -1 project", Computer Physics Communications, vol. 110, pp. 95-102, May 1998.

3 ATLAS DAQ Back-end Software User Requirements Document, ATLAS Internal Note DAQ-No-90, http://atddoc.cern.ch/Atlas/DaqSoft/document/draft_1.html.

4 High-Level Design of the ATLAS DAQ Back-end software. ATLAS Internal Note DAQ-No-87, http://atlasinfo.cern.ch/Atlas/documentation/notes/DAQTRIG/note87/DAQ_NOTE_87.ps.gz.

5 P. Croll et al., Use of Statecharts in the Modelling the Dynamic Behaviour of the ATLAS DAQ Prototype-1, IEEE Transactions on Nuclear Science, vol. 45, no. 4, pp. 1983-1988, August 1998.

6 P.Y. Duval et al., Test Report of the Run Control for the Atlas DAQ Prototype-1, ATLAS DAQ Prototype -1 Technical Note 113, http://atddoc.cern.ch/Atlas/Notes/113/Note113-1.html.

7 R. Jones, I. Soloviev, Configuration Databases in the ATLAS Prototype DAQ, CHEP98, Chicago USA, September 1998, http://www.hep.net/chep98/PDF/66.pdf.

8 R. Jones et al., The OKS Persistent In-memory Object Manager, IEEE Transactions on Nuclear Science, vol. 45, pp. 1958-1964, August 1998.

9 S. Kolos, I. Soloviev, Remote Database Access library: Users Guide, ATLAS DAQ Prototype -1 Technical Note 122, http://atddoc.cern.ch/Atlas/Notes/122/Note122-1.html.

10 I. Soloviev, Test Report of the Configuration Databases for the Atlas DAQ Prototype-1, ATLAS DAQ Prototype -1 Technical Note 114, http://atddoc.cern.ch/Atlas/Notes/114/Note114-1.html.

11 M.J. Carey et al., A Status Report on the OO7 OODBMS Benchmark Effort, Proceedings of OOPSLA94.

12 S. Kolos et al., Applications of CORBA in the ATLAS prototype DAQ, Proceedings of the IEEE Real Time Conference, Santa Fe, USA, June 1999.

13 D. Burckhart et al., Unit Test Report of the Message Reporting System for the Atlas DAQ Prototype-1, ATLAS DAQ Prototype -1 Technical Note 121, http://atddoc.cern.ch/ Atlas/Notes/121/Note121-1.html.

14 E. Badescu et al., Test Report of the Information Service for the Atlas DAQ Prototype -1, ATLAS DAQ Prototype -1 Technical Note 118, http://atddoc.cern.ch/ Atlas/Notes /118/Note118-1.html.

15 L. Tuura, Overview of ATLAS Software Release Tools, CHEP98, Chicago, USA, http://home.cern.ch /~lat/slides/98-36/.

16 D. Burckhart et al., Software technologies for a prototype ATLAS DAQ, Computer Physics Communications, vol. 110, pp. 113-119, May 1998.

17 A. Amorim et al., Use of Corba in the ATLAS Prototype DAQ, IEEE Transactions on Nuclear Science, vol. 45, no. 4, pp. 1978-1982, August 1998.

18 Tools.h++ Introduction and Reference Manual Version 6. Thomas Keffer, Rogue Wave Software Inc. 1994.

19 ILU documentation. Copyright (c) 1991-1999 Xerox Corporation. ftp://ftp.parc.xerox.com/pub/ilu/ilu.html.

20 CHSM: A Language System Extending C++ for Implementing Reactive Systems. P. J. Lucas, F. Riccardi. http://www.best.com/~pjl/software.html.

21 ACE. Douglas C. Schmidt. http://www.cs.wustl.edu/~schmidt.

22 CLIPS Basic Programming Guide, NASA JSC-25012

23 CmdLine:a C++ option-parsing framework,Brad Appleton http://enteract.com/~bradapp/ftp/src/libs/C++/CmdLine.html

24 I.Alexandrov et al., Performance and Scalability of the Back-end sub-system in the ATLAS DAQ/EF Prototype, Proceedings of the IEEE Real Time Conference, Santa Fe, USA, June 1999.

25 PY. Duval, R.Jones, K.Rybaltchenko, D. Schweiger, L.Tremblet, G.Unel, RCLDAQ: interface to the run control component, ATLAS DAQ Prototype -1 Technical Note 105, http://atddoc.cern.ch/Atlas/Notes/105/Note105-1.html.

26 H.J.Burckhart, M.Caprini and R.Jones, Connection DCS <==> DAQ in ATLAS, DCS-IWN8, 12 Nov 1999, http://atlasinfo.cern.ch/ATLAS/GROUPS/DAQTRIG/DCS/dcs_daq_0.6.pdf

27 E. Badescu et al., Component Integration Test Plan for the Back-end sub-system of the Atlas DAQ Prototype-1 , ATLAS DAQ Prototype -1 Technical Note 127, http://atddoc.cern.ch/Atlas/Notes/127/Note127-1.html.