# Performance and Scalability of the Back-end sub-system in the ATLAS DAQ/EF Prototype

I. Alexandrov[1], A. Amorim[2], E. Badescu[3], D. Burckhart[4], M. Caprini[3,4], L. Cohen[5], P.-Y. Duval[5], R. Hart[6], R. Jones[4], A. Kazarov[7], S. Kolos[4,7], V. Kotov[1], D. Laugier[5], L. Mapelli[4], L. Moneta[8], Z. Qian[5], A. Radu[3], C.A. Ribeiro[2], V. Roumiantsev[1], Y. Ryabov[7], D. Schweiger[4,9], I. Soloviev[4,7]

[1]Joint Institute for Nuclear Research, Dubna, Russia
[2]Lisbon Institute of Physics, Lisbon, Portugal
[3]Institute of Atomic Physics, Bucharest, Romania
[4]CERN, Geneva, Switzerland
[5]Centre de Physique des Particules, Marseille, France
[6]NIKHEF, Amsterdam, Netherlands
[7]Petersburg Nuclear Physics Institute (PNPI), Gatchina, St. Petersburg, Russia
[8]Section de Physique, Universite de Geneve, Geneva, Switzerland
[9]Institute for Experimental Physics, Innsbruk, Austria

## *Abstract*

The DAQ group of the future ATLAS experiment has developed a prototype system based on the Trigger/DAQ architecture described in the ATLAS Technical Proposal [1] to support studies of the full system functionality, architecture as well as available hardware and software technologies.

One sub-system of this prototype is the back-end which encompasses the software needed to configure, control and monitor the DAQ, but excludes the processing and transportation of physics data.

The back-end consists of a number of components including run control, configuration databases and message reporting system. The software has been developed using standard, external software technologies such as OO databases and CORBA. It has been ported to several C++ compilers and operating systems including Solaris, Linux, WNT and LynxOS.

This paper gives an overview of the back-end software, its performance, scalability and current status.

## I. INTRODUCTION

The ATLAS data acquisition (DAQ) and Event Filter (EF) prototype "-1" project [2] is intended to produce a prototype system representing a "full slice" of a DAQ suitable for evaluating candidate technologies and architectures for the final ATLAS DAQ system on the LHC accelerator at CERN. Within the prototype project, the back-end sub-system encompasses the software for configuring, controlling and monitoring the DAQ but specifically excludes the management, processing or transportation of physics data. The back-end software must co-exist and co-operate with the other sub-systems. In particular; interfaces are required to the triggers, processor farm, accelerator, event builder, Local DAQ (i.e. detector read-out crate controller) and Detector Control System (DCS).

It is expected that the operational environment for the back-end software will be a heterogeneous collection of workstations and PCs running WindowsNT or UNIX and embedded systems running various flavours of real-time UNIX operating systems (e.g. LynxOS) connected via a Local Area Network (LAN). Figure 1 shows on which processors the back-end software is expected to run, that is the event filter processors, supervisor and the LDAQ processors in the detector read-out crates.

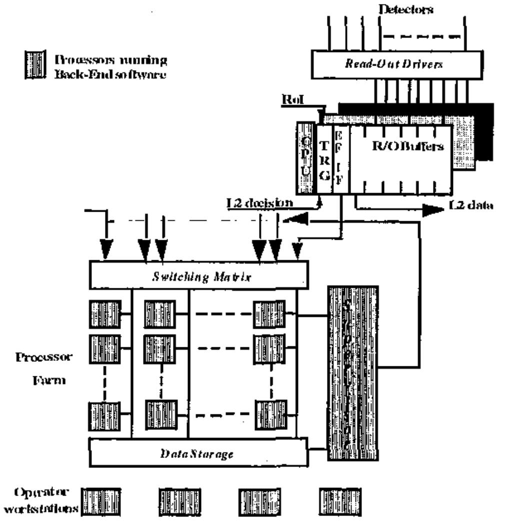

Figure 1: Back-end DAQ Operational Environment

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.

## II. BACK-END SOFTWARE COMPONENTS

### A. *The software component model*

The user requirements gathered for the back-end sub-system [3] have been divided into groups related to activities providing similar functionality. The groups have been further

0018-9499/00$10.00 © 2000 IEEE

developed into components of the back-end with a well defined purpose and boundaries. The components have interfaces with other components and external systems. Each component offers some unique functionality and has its own architecture.

The components have been grouped into two sets: core components and Trigger/DAQ and detector integration components. For all core components an implementation exists and functionality and performance tests have been performed. For most detector integration components the high level design is complete and an initial implementation is available. A short description of the components with some performance test results follows. For further information on the back-end components refer to [4].

## *B. Core components*

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

### *1) Run control*

The run control (RC) 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.

Due to the size and complexity of the DAQ, the run control component consists of many programs executing on several distributed computers connected through a network. Such a distributed system reflects the structure of the DAQ itself and is implemented as a hierarchy of entities called controllers, each with responsibility for a well-defined element of the DAQ system. The controller's state is the simplified external view of the current working condition of the element under its responsibility. The controllers are organized into a hierarchical tree structure that reflects the general organization of the DAQ system itself and is defined in the configuration database. The behaviour of each controller in the Run Control tree has been modelled using state charts. State charts reflecting the different aspects of a run controller's behaviour have been designed and subsequently implemented as C++ code using the CHSM language system [5]. The controller skeleton is customised using C++ inheritance by developers of individual controllers in the tree by adding code to implement the required functionality.

An important task of the DAQ back-end software is to marshal the DAQ through its start-up and shutdown procedures so that they are performed in an orderly manner. The DAQ Supervisor program encompasses this functionality. It is responsible for the creation of all software processes (including all the controllers) during start-up of the DAQ according to the configuration defined in the database. It uses the Process Manager to create processes on the processors of the DAQ system. It is also responsible for ensuring the orderly shut-down of the DAQ system at the end of data taking activities.

In order to determine the overhead of using the run control to marshal data taking activities, a suite of tests have been defined, giving the possibility to measure the "cold start" time (time it takes to start all controllers, associated servers and do all necessary transitions to get to the Running state), "lukewarm start" time (assuming all the controllers are started and in the Initial state, time to get to the Running state), "warm start" time (time to start a run with the same configuration) and the corresponding "stop" times. Different configurations have been tested [6], with up to 250 controllers in the tree and up to 10 levels in the hierarchy. Some results are presented in the Table 1.

Table 1
Run Control scalability results (time in seconds averaged over 10 cycles)

| Controller Number | 11 | 51 | 251 |
|---|---|---|---|
| cold start | 2.3 | 6.0 | 28.0 |
| cold stop | 1.5 | 4.0 | 18.0 |
| lukewarm start/stop | 0.5 | 1.0 | 3.0 |
| warm start/stop | 0.2 | 0.4 | 1.0 |

The test results show that, even for the largest configuration expected, the time to change the state of the controller tree is in an acceptable range (maximum 1s per command).

### *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 status. One of the major design issues of ATLAS DAQ is to be as flexible as possible and so the software is parameterized by the contents of the configuration database.

After an evaluation of various commercial and shareware persistence systems (relational databases, object databases and object managers), it was decided to adopt a two-tier architecture, using a light-weight in-memory persistent object manager to support the real-time requirements and a full Object Database Management System (ODBMS) as a back-up and for long term data management [7]. For the run-time object manager, a package called OKS (Object Kernel Support) [8] has been developed on top of Rogue Wave's Tools.h++ C++ class library. The OKS system stores database schema and data in portable ASCII files and allows different schema and data files to be merged in a single database. It includes Motif based GUIs to design database schema and to manipulate OKS objects. For the back-up and for long-term data management, a commercial object oriented database management system Objectivity/DB has been chosen by the experiment.

Data access libraries (DAL) are used to hide the details of the low-level persistent object system from a programmer. Three DALs have been implemented for the DAQ configuration database: "data flow application view" (maps the database object schema to the data flow application view), "back-end application view" and an automatically generated C++ DAL (maps database schema to C++ classes and database objects to C++ objects with get...()/set...() methods).

In order to provide access to the configuration databases from remote applications or from applications written in a language other than C++, a CORBA interface has been implemented. The clients (C, C++, Java, etc.) access, through an IDL interface, a remote server written in C++ [9].

Tests have been performed on the configuration databases to evaluate the performance for different configurations (including expected DAQ configurations), to test compatibility of OKS with commercial persistent object managers and to assess it's reliability [10]. An independent performance and functionality benchmark [11] has been used to provide a comprehensive profile. The tests have been made for different configurations, where the number of objects varies from 5,000 to 1,000,000. The benchmark shows that OKS requires less system resources when working with relatively small configurations. For the biggest configuration, the sizes of data files are comparable, but OKS requires more time and memory for initialization (OKS is an in-memory database). OKS shows better performance of read-only operations for configurations of any size: the transversals are faster by about 10 times, the queries are faster by up to 100 times (1000 times for path lookup). The results of the benchmark show that OKS is preferred when working with small configurations (which are close to the expected DAQ configurations) and from this point of view is better for run-time usage. The Objectivity/DB works better with large databases (which correspond to multiple DAQ configurations including versioning) in an off- line environment.

Another benchmark evaluates the performances for various expected DAQ configurations upto the final ATLAS DAQ (200 crates and ~100,000 database objects) on different computers running various operating systems (Sun Solaris, HP-UX, LynxOS and Windows NT) and shows what are the hardware requirements to achieve acceptable response times. This benchmark has been performed with OKS only (and not Objectivity/DB) because it is the run-time object manager and hence directly affects the DAQ performance during data taking activities. By the benchmark's results the fastest computers require about one second for initialization of a medium-sized configuration (i.e. prototype-1 DAQ) and about half of one second for it's complete traversal and shut-down. The average front-end computer (100 MHz PowerPC with 32 MB running real-time LynxOS) requires about 3,000 ms to initialise the database and load a medium-sized configuration.

The two-tier architecture helps cover all requirements for the configuration databases (high performance, availability for real-time operating system running embedded processors, data and schema versioning) and eases data exchange with off-line applications. The performed tests and benchmarks show robustness and compatibility of OKS and Objectivity/DB.

### *3) Message reporting system*

The Message Reporting System (MRS) allows software components in the ATLAS on-line system to report and receive error messages. The MRS performs the transport, filtering and routing of messages.

The MRS has a client - server architecture and is based on the IPC package, which provides the basic functionality for working with partitions and assures an interface to a CORBA implementation [12]. The server interface is described in IDL and provides the message reporting, the filtering and subscription mechanism. The clients can be senders and/or receivers of messages. A message is composed at the sender side as a data stream and contains the name, the severity level, a message text, some qualifiers (machine name, process id, a time stamp, other optional qualifiers) and some optional parameters. Messages can be further defined in a message database (implemented in OKS), which the server uses to complete the message during the reporting process. The receivers can subscribe for groups of messages defined by a subscription criteria (a logical expression containing message names, severity levels and optional qualifiers). If a reported message fulfils the subscription criteria, the subscribing receivers are notified by a callback mechanism.

Scalability and performance tests for MRS provide information about the transport time of a message from a sender to a receiver under a chosen set of test conditions. A large number of tests with a varying number of senders and receivers have been performed on single platform systems (Solaris, HP-UX, LynxOS, WindowsNT and Linux) as well as on distributed configurations with combinations of the mentioned platforms [13]. Table 2 presents the results from the performance tests with 1, 10 and 50 senders and 1, 5 and 10 receivers on a distributed configuration including a server on a Solaris machine and senders and receivers on HP-UX and LynxOS machines. The results indicate the mean transfer time (in milliseconds) of one MRS message from one sender to one of the receivers.

Table 2
Message Reporting System: Reporting time per message (ms/msg) for a distributed configuration

| **Number of senders =>** | **1** | **10** | **50** |
|---|---|---|---|
| 1 receiver | 6.0 | 2.5 | 3.1 |
| 5 receivers | 8.0 | 6.6 | 6.6 |
| 10 receivers | 12.0 | 9.5 | 10.6 |

The capacity of the component can be quantified as the maximum number of received messages per second. For a distributed system with 50 senders, it is approximately 325 messages for a single receiver, 150 messages for 5 receivers and 100 messages for 10 receivers.

### *4) 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 request.

The IS has a multi-server architecture and uses also the IPC package as an interface to a CORBA implementation. An information is an object having a name (a string identifier which must be unique across all other objects in one server), a type (a string identifier of a particular type of information), a value and a time stamp. Information objects are stored on a server which must have a unique name across all the servers in a particular partition. Any client application can access any information in any server of any partition. It can create, update or delete an information of any type, retrieve the value of an information or obtain the list of all information existing on a server. A subscription mechanism allows a client to subscribe to a particular information object or group of information objects and be notified when theinformation is changed. A server can backup and restore all the information objects and all the subscriptions to a file for check pointing purposes.

The multi-server architecture gives the possibility to have different servers for different DAQ domains and assures scalability. Tests have been performed to evaluate the performance of the IS and the capability of the system to work with multiple information objects and multiple sources [14]. The tests measure the mean information publishing time, the mean update and notify cycle and their dependence upon the number of sources and receivers as well as the size of information, on the same machine and for distributed configurations. Three types of information objects have been used: "large" with 34 fields (most of them arrays), "medium" with 10 fields and "small" with only one field. The configurations chosen for these tests were all possible combinations of 1, 5 and 10 receivers and 1, 10 and 50 sources working in the same partition. The test conditions reproduced as closely as possible the true operational environment for the Information Service (servers on workstations, multiple sources and receivers on the front-end machines and on the workstations). Table 3 presents the results of the tests on a distributed configuration (server on a Solaris machine and sources and receivers on HP-UX and LynxOS machines) for the whole information cycle (from source to receiver) for the update operation (in milliseconds) for large, medium and small information sizes.

Table 3
Information Service: Mean values of the updating time for a distributed configuration (in ms)

| **Number of Sources =>** | | **1** | **10** | **50** |
|---|---|---|---|---|
| **large** | 1 receiver | 11.0 | 4.6 | 3.5 |
| | 5 receivers | 20.0 | 14.5 | 4.5 |
| | 10 receivers | 35.0 | 28.3 | 13.6 |
| **medium** | 1 receiver | 4.0 | 1.6 | 1.3 |
| | 5 receivers | 5.0 | 4.3 | 2.5 |
| | 10 receivers | 8.5 | 11.2 | 9.0 |
| **small** | 1 receiver | 3.5 | 1.2 | 0.8 |
| | 5 receivers | 5.0 | 3.7 | 2.4 |
| | 10 receivers | 8.5 | 10.4 | 5.8 |

It is expected that most of the information used in the data acquisition system will have several fields (medium size). For such information, on one server with 50 sources and 5 receivers it is possible to have up to 400 information updates per second.

### *5) Process manager*

The Process Manager (PMG) performs basic job control of the DAQ software components. It is capable of starting, stopping and monitoring the status (e.g. running or exited) of software components on the DAQ processors independently of the underlying operating system.

The process manager service consists of three main elements: agents, clients and the dynamic database. The purpose of an agent is to manage processes on a single computer host. Each computer participating in the DAQ activities on which the services of the process manager is required must run an agent. The agents wait for client requests and create or kill processes, give the updated information to the dynamic database, check periodically the processes states and call the concerned clients when something happens to their processes. The dynamic database stores the data about the created processes and the running agents of the PMG and is implemented as an Information Service server. The PMG provides a C++ client interface as a library that uses the configuration database to retrieve details of programs it is requested to start. The primary client of the PMG is the DAQ Supervisor program described above.

In order to test the scalability and performance of the process manager, the time to create and destroy a process in various circumstances has been measured. The process creation and destruction time have an effect on the DAQ system availability when changing configuration or starting a new data taking session. The time measured represents the delay between a process manager client making the series of requests and receiving confirmation that the operations have been completed. Initial measurements show that the delay in creating a process using the process manager increases slowly with the number of processes managed. For 100 processes per agent the delay is less than 200 ms. For process destruction a delay of less than 100 ms was obtained. The tests have been made with clients and agents working in a distributed configuration.

## *C. Trigger / DAQ and detector integration components*

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

### *1) 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 Resource Manager (RM) is needed to formalise the allocation of DAQ resources and allow groups to work in parallel without interference.

The RM is a separate server to which clients make requests to reserve and release resource tokens. Tokens represent the dynamic state (i.e. allocated or available) of limited resources. The RM controls tokens but not the access to the associated resources themselves. It is the responsibility of applications (directly or indirectly via the process manager) to ensure they do not access limited resources without first acquiring the necessary tokens. The RM_Client class implements the client part of the RM service and is the interface between user programs and the RM.

### *2) On-line bookkeeper*

The on-line bookkeeper (OBK) archives 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.

OBK includes a process that listens for DAQ messages, either from the Information Service or the Message Reporting System, and stores the information on the Bookkeeper database. Users may add their own messages to the database via the interface.

### *3) Test manager*

The Test Manager (TM) organises individual tests for hardware and software components. The individual tests themselves are not the responsibility of the test manager which simply assures their execution (via the process manager) and verifies their output. The individual tests are intended to verify the functionality of a given component.

The test manager consists of two main parts: a client class, which is the interface to the user and a repository class, which contains a set of tests. The TM uses the Process Manager to start and stop a test and the Configuration Database to store the descriptions the tests.

### *4) Integrated graphical user interface*

The integrated graphical user interface (IGUI) gives a view of the status of the data acquisition system and allows the user to control its operation. The IGUI is intended mainly for general users, such as a shift operator at a test beam, but it also provides functionality for DAQ experts to control and debug the DAQ system.

A prototype version of the IGUI written in Java (JDK 1.2) communicates with other components through Java IDL. The IDL definitions used for the component C++ interfaces have been reused without modification. The user can set the run parameters and send commands to the DAQ supervisor (to boot or shutdown the DAQ configuration) and to Run Control (to start data taking activities). In different panels the user can see the run control tree and the controllers status, the list of PMG agents and running processes, received MRS messages as well as informations about other sub-systems (data flow, event filter) via the IS.

### *5) Diagnostics package*

The diagnostics package uses 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.

## III. INTEGRATION AND VALIDATION

Each component was 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) as shown in Figure 2. 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.

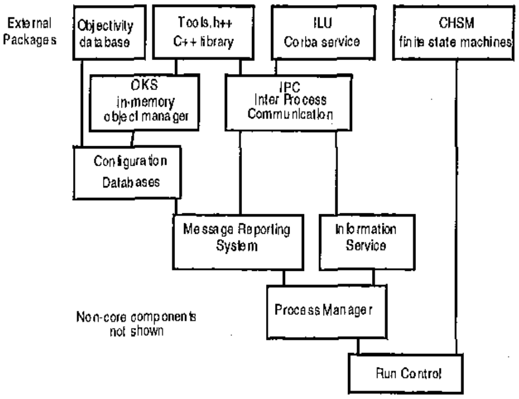

Figure 2: Dependencies between the core components and principal external packages

The integration tests simulate the control and configuration of data taking sessions. This requires completing the content of the configuration database to define all the necessary software elements. Processes are launched by the DAQ supervisor (via the process manager) and the run control marshals the hierarchy of controllers from an Initial to Running state then back again to represent a single data taking run. Information for various components is made available in the Information Service and error messages are distributed to the operator displays and log files using the Message Reporting System. The cycle is repeated to represent a full data taking session.

During the tests the concept of partitions (independent entities that can be used to take data or perform tests) proved very useful. An implementation of partitioning using CORBA Naming Service was included in the IPC package [12] and used by all back-end components, giving the developers the possibility to run tests in parallel. Component specific developer interfaces (in Motif) and the integrated graphical user interface (in Java) have been used during the tests.

A configuration management system based on the ATLAS Software Release Tools [15] and integrated with testing tools to produce static software metrics (Logiscope) and code coverage measurements (Insure++) has been used. The configuration management policy is to build a new release every month and to use a nightly build for development and testing purposes.

## IV. Summary and Future

The unit and integration tests described above are intended to provide a first 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 test scalability for the final ATLAS DAQ system and 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 supervisor's 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 experiment's start-up (2005).

## V. Acknowledgments

We would like to thank our computer system managers, Corine Costaz and Tony Wildish.

## VI. 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", CHEP'98, 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/Note 122-1.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 OOPSLA'94.

[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", CHEP'98, Chicago, USA, September 1998, 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.