
Studies on online book-keeping for the ATLAS experiment

Sunderland University

Dissertation for the obtention of the M.Sc. degree in Computing

author: **Levi Pedro Silva Lucio**

from Portugal

Distance learning student based at CERN, Geneva

Work supervised by:

Dr. Norman Parrington, University of Sunderland, UK

Dr. Robert Jones, CERN, Switzerland

Geneva, September 2002

Abstract

ATLAS will be one of the four detectors of the LHC (Large Hadron Collider) particle accelerator currently being built at CERN, Geneva. The project is expected to start production in 2006 and during its lifetime (15-20 years) to generate roughly one petabyte per year of physics data. This vast amount of information will require a powerful and scalable tool to do the online acquisition and offline management of log books about things such as what parameters the detector used while acquiring a certain set of data, how the physical conditions of the detector changed during acquisition, how the data is logically and physically organized or what errors occurred during data taking.

This document presents studies on such a tool from the technological and design points of view. After the definition of the problem, three implementations using three distinct DBMS systems (Objectivity/DB, OKS and MySQL) are presented, tested, discussed and compared.

Acknowledgements

I'd like to thank Norman Parrington and Bob Jones for the guidance and for helping me to stay on a clear path from beginning to end during these two year's work.

Many thanks are also in order to Livio Mapelli, who made this project possible and who always tried his best to provide the necessary resources for the work to go on.

My gratitude also goes to the Lisbon ATLAS-DAQ team, with whom I worked to develop the several OBK prototypes. Thanks to Luis Pedro and Andre Ribeiro for their work on implementing the MySQL and Objectivity/DB OBK prototypes. Thanks to Mario Monteiro for his work on the OBK/MySQL web-based browser. My appreciation goes also to Antonio Amorim for some of the ideas that went into this dissertation.

I'd also like to thank the ATLAS Online Software team for the warm and efficient working atmosphere. Special thanks to Mihai Caprini for his availability and for letting me the time to work on this project. Thanks to Monika Barczyk for the development of the OBK/OKS web-based browser and to Igor Soloviev and Serguei Kolos for their help on using the OKS and the IS systems.

A big hug to my friends here in Geneva who made these two years a very special period of my life and with whom I share my sadness and my joys. I would also like to thank my friend Vasco Amaral for his suggestions about this work and many other things.

Thanks to Cinzia Borel-Messerli for helping me when I most needed it.

Dedico este trabalho ao meu pai, à minha mãe e ao meu irmão.

Table of contents

CHAPTER 1	<i>Introduction</i>	<i>1</i>
	1.1. Overview	2
	1.1.1. The physics	2
	1.1.2. The machinery	3
	1.1.3. The triggers and data acquisition system	6
	1.1.4. Offline computing	8
	1.2. MSc program objectives	9
	1.3. Dissertation's structure	10
	1.4. Summary	11
CHAPTER 2	<i>Scope of the problem</i>	<i>13</i>
	2.1. High Level Requirements	14
	2.2. Previous Investigation	20
	2.2.1. Use of OO	20
	2.2.2. Objectivity/DB experience	21
	2.3. Structure of the problem	24
	2.3.1. Research into DBMS	24
	2.3.2. Software developing environment and process	25
	2.3.3. Evaluation	26
	2.4. Summary	28
CHAPTER 3	<i>Databases (in High Energy Physics)</i>	<i>29</i>
	3.1. Database concepts	30
	3.1.1. Database principles	30
	3.1.2. Database technology evolution	32
	3.1.3. Current picture	36
	3.2. Databases and HEP	38
	3.2.1. Some history	38
	3.2.2. Current trends	39

3.2.3. Current problems	41
3.3. Summary	43

CHAPTER 4

Physics' metadata gathering for ATLAS' Online Software 45

4.1. The Online Software from the OBK's viewpoint	46
4.1.1. Communication infrastructure	47
4.1.1.1. MRS	48
4.1.1.2. IS	50
4.1.2. Configuration Databases	51
4.2. The OBK as an Online Software component	53
4.2.1. Requirements	54
4.2.1.1. Use Cases	54
4.2.1.2. Assumptions and Dependencies	55
4.2.1.3. Constraints	55
4.2.2. Conceptual view of the OBK system	56
4.3. Summary	57

CHAPTER 5

The OBK software 59

5.1. The OBK from the user's viewpoint	60
5.1.1. Data Acquisition	60
5.1.1.1. Logical structure of the OBK database	60
5.1.1.2. Operating system environment	62
5.1.1.3. Starting and stopping the acquisition	63
5.1.1.4. Annotating book-kept data	66
5.1.2. Data browsing	69
5.1.2.1. Command line utilities	69
5.1.2.2. Web-based browser	70
5.1.2.3. C++ query library	72
5.1.3. Utilities	73
5.1.4. Available functionality per prototype	75
5.2. Implementation issues	76
5.2.1. Languages and tools	76
5.2.2. Objectivity/DB prototype	77
5.2.2.1. Web browser implementation	83
5.2.2.2. OBK/Objy Specificities	84

	5.2.3. <i>OKS prototype</i>	86
	5.2.3.1. Web browser implementation	92
	5.2.3.2. OBK/OKS specificities	92
	5.2.4. <i>MySQL prototype</i>	93
	5.3.3.1. Web browser implementation	98
	5.3.3.2. OBK/MySQL specificities	98
	5.2.5. <i>Supported platforms</i>	100
	5.3. Summary	102
CHAPTER 6	<i>Evaluation of the practical work</i>	103
	6.1. Test and integration phase results	104
	6.1.1. <i>Functionality and error recovery tests</i>	104
	6.1.2. <i>Performance and scalability tests</i>	105
	6.1.2.1. Test definition	105
	6.1.2.2. Test results	106
	6.1.2.3. Test results discussion	109
	6.1.3. <i>Connectivity to other Online Software components</i>	112
	6.2. Deployment	113
	6.2.1. <i>Large scale tests (of the Online Software)</i>	113
	6.2.2. <i>Testbeams</i>	113
	6.3. Some metrics	116
	6.4. Lessons learnt and practical recommendations	118
	6.5. Summary	122
CHAPTER 7	<i>Evaluation of the research</i>	125
	7.1. DBMS technology for the OBK	126
	7.1.1. <i>Pointers from the developed work</i>	126
	7.1.2. <i>Pointers from the database community</i>	126
	7.1.3. <i>Recommendations</i>	127
	7.2. Related work	129
	7.3. Future work	130
	7.4. Summary	131

Glossary **133**

References **137**

APPENDIX A

*List of binaries, libraries and scripts for each
OBK prototype* **145**

A.1. OBK/Objy **146**

A.2. OBK/OKS **148**

A.3. OBK/MySQL **150**

As indicated by the name, Chapter 1 provides an introduction to the present dissertation.

In section 1.1. a brief overview of CERN, particle accelerators and particle detectors is provided. The goal is to locate the OBK project into the larger scope of experimental particle physics.

Section 1.2. establishes both the working and the research objectives for the MSc program described in this dissertation.

Finally in 1.3. the document's structure is laid out.

1.1. Overview

The work described in this dissertation was developed in the context of the ATLAS (A Toroidal LHC ApparatuS) experiment, based at CERN (European Organization for Nuclear Physics).

Founded in 1954, CERN is a multinational collaboration having as goal the empirical study of physics, more precisely the study of matter and the forces that hold it together. To this effect several generations of increasingly powerful machines were built in order to probe deeper and deeper into the basic constituents of matter. Within its lifetime, several important physics discoveries have been made at CERN, along with revolutionary technology products such as the World Wide Web (WWW).

At the present moment, the next generation of matter probing machines is being built, expected to be ready for 2007. From the technology usage and complexity points of view, these machines are amongst the most complex devices ever built. In the several ongoing projects at CERN one can find a very colorful interdisciplinary blend in which computing has a very large role to play.

1.1.1. The physics

Although not directly in the scope of this dissertation, it is appropriate to discuss the sort of physics on which CERN concentrates. The functioning principles of the machines CERN builds will also be addressed.

As was already mentioned, CERN focuses on studying particle physics, i.e., understanding what are the basic components of matter, how they behave and what are the forces that keep them together. This is a difficult task given the successive barriers that appear while trying to understand the physics microcosm. The problem goes as follows: CERN machines are able to look inside matter by literally “smashing” (in physicist jargon “colliding”) accelerated particles - electrons, protons, etc. - and looking at the snapshots of the results of those collisions. These

results are more interesting as the collision is more energetic - given Einstein's well-known principle $E=mc^2$ (c being the speed of light in vacuum) the higher the energy, the more matter it can convert into. This way, the more energy one can provide a particle by accelerating it, the more matter will be produced when it collides. The more matter produced, the higher the probability of making rare and heavy particles appear.

Clearly it is not possible to provide unlimited quantities of energy to a particle, in order to accelerate it as close as possible to the speed of light. The amount of energy is thus one of the factors that limits the advance of particle physics and poses major challenges to the HEP community - HEP meaning High Energy Physics, for obvious reasons.

One of the major tasks foreseen for the new generation of machines being built at CERN is to discover a particle that, if found, will complete the fundamental standard model that explains matter. This particle is called the Higgs boson and is expected to be produced at the energies that will be made available by the LHC. If it exists, this boson will explain the mechanism by which matter acquires mass.

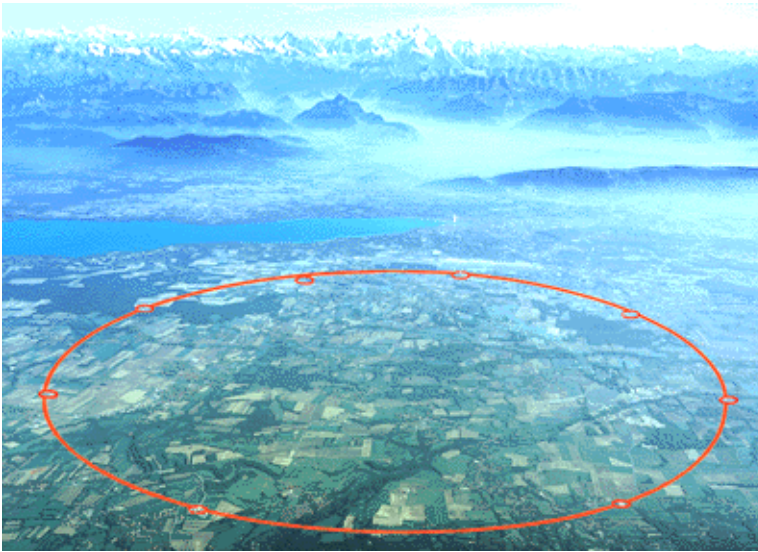
1.1.2. The machinery

In order to accelerate particles to such high energies and be able to measure the results of the collisions, a vast amount of machinery is being built at CERN. This machinery can be generically divided into: accelerator facilities; detector facilities.

As the name indicates, the accelerator is the piece of hardware responsible for providing energy to the particles, taking them to speeds close to that of light. This machine, called the LHC (Large Hadron Collider), is a huge ring (see fig. 1.1.)

measuring 28 Km in circumference. Given the fact that such a large construction would interfere with surface life, the LHC is built underground.

FIGURE 1.1. LHC aerial view



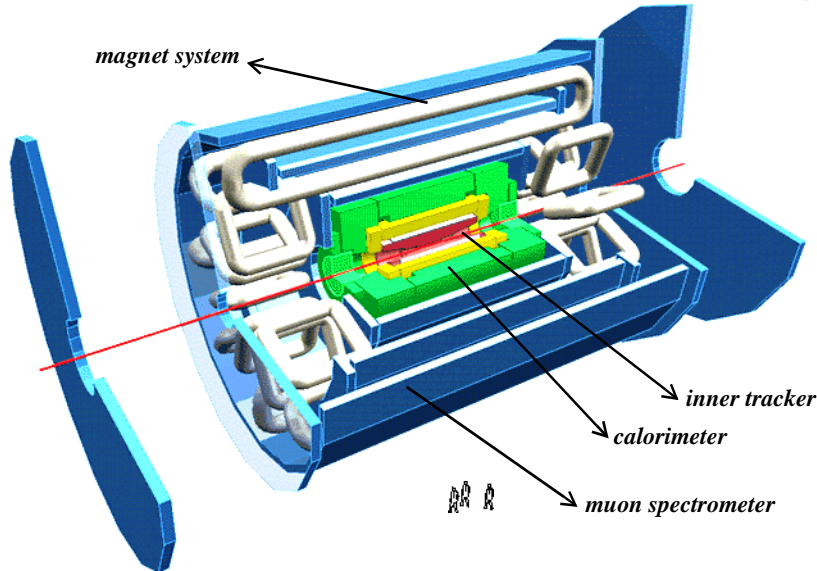
There are two types of accelerators - linear and circular. They both use the same principle: electric fields provide energy to accelerate the particles and magnetic fields to keep them on a predefined track. The difference between linear and circular accelerators consists in the fact that it is possible to inject progressively more energy in a particle circulating inside a ring, while for a linear accelerator the length of the machine necessarily limits the obtainable acceleration.

Along the LHC accelerator ring four detectors are being put in place - one of which is ATLAS. As large as a five story building, the ATLAS detector is being built by an international collaboration of 2000 physicists from 34 countries. As already discussed, in order to study the interior of matter it is necessary to make small particles collide at very high energies. There are however two different types of approaches used to provoke these collisions: fixed target or colliding beams. While in fixed target detectors particles are made to crash into a solid body (usu-

ally wires), the detectors which use colliding beams make two bunches of particles traveling into opposite directions meet at a certain point in space. There are obviously advantages and drawbacks for each solution: while the collision rate is much higher in the fixed target experiments, the energy absorbed by the target itself makes the production of new particles lower than in the case of colliding beams. For the ATLAS experiment colliding beams are used.

After a collision happens, it becomes necessary to detect and measure the results (the whole of the results of the collision is called an *event*). The newly created particles have such short lifetimes that only the decay results can be observed by the hardware. This makes it necessary for the detector to be a very sensitive machine, able to react very quickly to a wide range of different particles. The ATLAS detector consists of several layers (see fig. 1.2.), each of them having a specific function: the inner tracker is used for measuring the momentum of charged particles; the calorimeter to measure particle energy; the muon spectrometer to identify muons. A magnet system is also built-in to provide a magnetic field that will bend charged particles in order to measure their momentum.

FIGURE 1.2. The ATLAS detector



1.1.3. The triggers and data acquisition system

Since the physics events looked for are very rare, the experiment will have to be repeated until statistics which validate the theory are available. The rate of particle collisions inside the detector will thus be extremely high - around 40 million per second; on the other side, only a small proportion is actually interesting and worth storing¹. This trend implies a very powerful filter mechanism that will select, transport and store the few remaining events (100 per second - 200MBytes) that contain potentially interesting information.

Also, given the complexity of the detector, it is necessary to constantly monitor its condition. In order to be able to have correct measurements for the produced events, the understanding of the calibration and alignment of the enormous amount

1. In fact, given the current technology, it would be impossible to store all the 40 million events per second. The transport and storage speeds required would be orders of magnitude above what is possible today.

of detector pieces is absolutely indispensable. In parallel with the event filtering mentioned in the previous paragraph, the data concerning the state of the detector is also being constantly read out and used to keep all the systems in working condition.

To address the event selection problem, a very sophisticated trigger (filter) and data acquisition system is being built for ATLAS (ATLAS TDAQ). This system consists of three levels of triggering, each one elaborating on the previous one's decisions. At the first level (LVL1), special purpose processors make decisions about the interest of an event in a time frame of 2 microseconds, reducing the initial 40MHz rate (2 ns between events) to 100KHz. In order to be that fast the first selection is obviously very rough, looking only at very small parts of the full event.

The second level trigger (LVL2) inputs the 100 KHz rate remaining from the first level analysis and reduces it to 1 KHz by evaluating larger assembled parts of the full event (from the various sub-detector components of ATLAS).

Finally the third level trigger looks at the fully assembled events (taking into consideration all the detector information) remaining from the LVL2 and, by applying complex selection algorithms, reduces the 1KHz rate to 100Hz. Being that each event occupies 2Mbytes in permanent storage, the final recording rate is in the order of 200 MBytes per second.

Obviously, to keep the detector and the triggers working in an orderly fashion several systems providing services such as, detector read-out and buffering, data transportation, control, configuration, monitoring, or synchronization are necessary. The set of systems providing these services is generically called data acquisition system² (DAQ).

The work described in this dissertation is part of a DAQ sub-system - the Online Software, responsible for providing control, configuration and monitoring services

2. This is a simplified account given that there is no clear border between the third level trigger and the DAQ. Note that this dissertation is a snapshot of the state of development of the ATLAS experiment at a given point of time. Further changes should be expected before the experiment is fully operational.

to the detector and the TDAQ. More precisely the dissertation has its focus on the Online Book-keeper (OBK), the tool deployed to store and manage log information for both the detector and all of the TDAQ. This log information is varied and encompasses whatever the detector or TDAQ system may want to keep track of.

1.1.4. Offline computing

The systems described in the previous section have one common characteristic - they are working online with the detector, thus real-time constrained by the need of handling the extremely high rate of flowing physics information. However, once the interesting events are permanently stored further analysis can be performed on them offline, the time constraints being non-existent.

The three-level trigger system provides a first level of analysis and selection of physics data, which is necessarily handled in an automatic fashion. Offline, the humans trying to make sense out of the recorded data will look at the events using methods supported by a software and hardware framework. This framework is being built in parallel with the detector and the TDAQ system. Although the OBK data will be used offline to support the physics data analysis phase, the study of the offline software is beyond the scope of the present text.

1.2. MSc program objectives

The full description of the objectives for the undertaken MSc program can be found in appendix B, the *learning contract* document. Being an industrial practice distant MSc, the practical component of the work is obviously prominent. From CERN's point of view, a functional Online Book-keeper had to be implemented, documented and deployed during the learning time.

From the research viewpoint it was necessary to study a specific discipline of computing, produce valid results while applying current state-of-the-art technology in that area and match the obtained results with existing theories. Clearly it was necessary to blend the practical and the theoretical activities in a way that both were correctly balanced and could interact in a productive way. This dissertation tries to explore that interaction and produce practical results which make use of current theories. Also, the presented theoretical considerations are in part deduced from the experience and insights acquired while building the required software.

Taking into consideration the previous paragraphs, one can thus define the objectives of the current dissertation as being able to, in a sustained way, recommend adequate resources for a Book-keeper to be included in ATLAS' Online Software system. The OBK can be seen as a database tool, which justifies the choice of *database systems* as the theoretical research topic. It is also important to mention that the OBK works online with the detector, which implies some (soft) real-time constraints.

1.3. Dissertation's structure

The present document is composed of seven chapters, in the following order:

- **Chapter 1** is the introduction. The context of the ATLAS experiment is defined in order to lay the ground for understanding the addressed book-keeping problem and set goals for the MSc work;
- In **Chapter 2** the scope of the problem is enlarged and its several aspects are laid out in order to make possible the definition of a working methodology;
- **Chapter 3** is the dissertation' theoretical section, exploring database systems theory as well as the usage of database systems in High Energy Physics;
- **Chapter 4** starts by exploring the several Online Software components the OBK interacts with. It then defines the high level requirements and a conceptual model for the book-keeper tool;
- In **Chapter 5** the OBK software is described in-depth, both from the implementor's and the user's point of view;
- **Chapter 6** is an evaluation of the practical work developed throughout study time. Topics such as functionality, performance or scalability are analyzed for each of the built OBK prototypes. Some metrics concerning the used methodology and software development process are also provided. These are complemented by a relation between the proposed learning contract (appendix B) and the objectives which were actually fulfilled. The chapter ends with a "lessons learnt" section including recommendations for an architecture for the OBK;
- Finally, **Chapter 7** elaborates on the previous chapter to draw considerations about the theoretical aspects of the work. As was defined in the dissertation' objectives, recommendations regarding and technology to build a production OBK system are provided.

1.4. Summary

CERN is a multinational collaboration having as it's goal the study of particle physics. For that purpose a new generation of machines is being built, among which is the ATLAS detector. ATLAS is one of the detectors of the LHC accelerator that will be used to measure the physics events that will be produced when two opposite particle beams meet.

Since the sought after physics events which are being searched for are very rare, the experiment is repeated at a vertiginous rate: one event every ns. Since only a small fraction of these events are interesting, a powerful filter is necessary in order to discard uninteresting information. This filter is composed of three levels, each one of them elaborating on the decisions made by the previous one. In order to provide control, configuration and monitoring to this filter as well as to the detector, a software system called Online Software is being built.

The work presented in this dissertation consists of the studies performed in order to recommend adequate resources to implement and deploy a book-keeper for ATLAS' Online Software. This tool called OBK (Online Book-Keeper) will store and manage log information for ATLAS' systems that will require those kind of services. The dissertation focuses on the two main aspects of the work: the implementation and deployment of the several OBK study prototypes; the research done on the tools necessary to build the OBK, i.e., Database Management Systems, as well as on software engineering processes and tools adequate to the project.

This chapter provides an abstract description of the generic problem tackled in this dissertation - to recommend adequate resources for a book-keeping production system for ATLAS's Online Software.

As it involves a considerable number of variables, the proposed solution is to divide the problem into several domains of work, which may be sequential or not, and to consistently merge the acquired knowledge in the end.

Chapter 2 starts by putting the generic problem into perspective in 2.1 (High Level Requirements) and 2.2 (Previous Investigation). In 2.3 (Structure of the problem) it goes on to describe the proposed sub-domains of work and how they fit together.

2.1. High Level Requirements

As mentioned in the introduction, the Online Book-keeper is a software component of ATLAS's Online Software [1], which itself is a sub-system of the Trigger-DAQ (TDAQ) - the set of software and hardware systems responsible generically for acquiring, filtering and moving data from the detector to mass storage, in order to make it available for later analysis. The data dealt with are what the physicists call "events", which unitarily describes the interaction of two particles and the final state products of that interaction. Since the event rate is very high and only some of these events are actually interesting, a powerful filter is needed.

Generically speaking, the Online Software¹ is the system keeping all the other components of the Trigger-DAQ working in a coordinated fashion, by providing configuration, control and monitoring services to a range of other systems. These systems are:

- **Level 1 Trigger (LVL1):** First level of triggering. Informs the following levels of the system that a certain set of signals is potentially interesting;
- **Detector:** The interesting data is extracted out of the detector by means of RODs (Read Out Drivers). These are specialized boards connected directly to the detector;
- **DataFlow:** Responsible for moving the event data from the detector readout links to the final mass storage;
- **DCS (Detector Control System):** Doesn't deal with physics data directly, but ensures a standard and secure hardware environment for the safe acquisition of physics data;

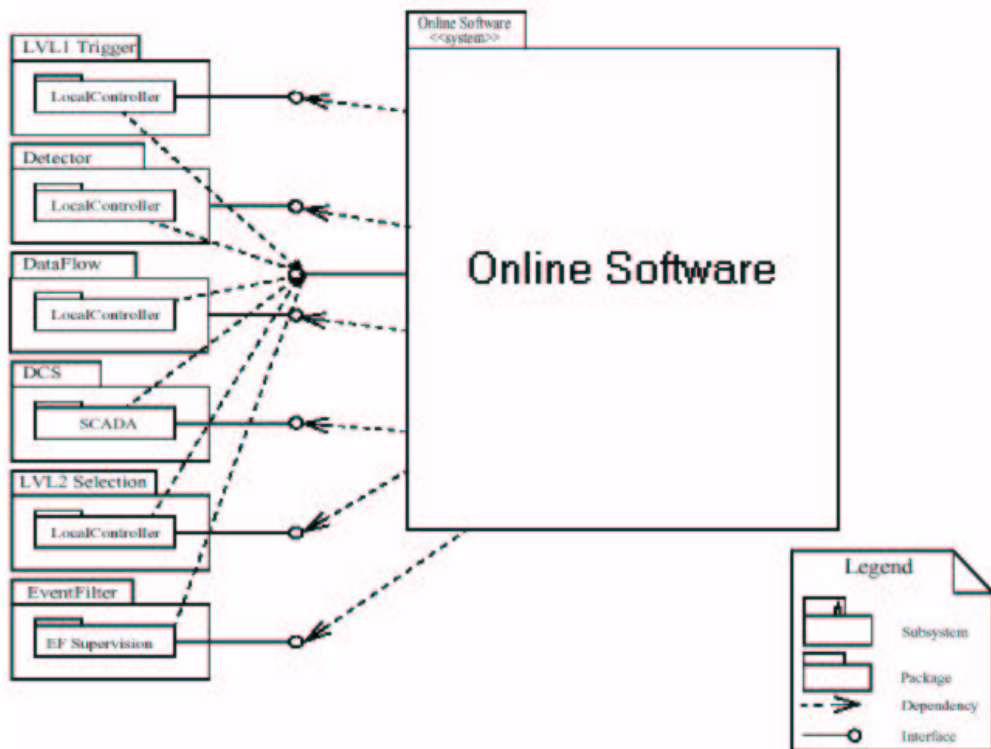
1. In the framework of the DAQ/EF-1 prototype project the Online Software was called Back-End DAQ System. The DAQ/EF-1 was a previous project aimed at implementing a "*full vertical slice of the functional DAQ and EF architecture*". The goal was to provide a test-bed for technological and architectural testing and refining.

2.1. High Level Requirements

- **Level 2 Trigger (LVL2):** First part of the High Level Trigger (HLT) system. Analyses previously selected data by LVL1 and provides another level of filtering of data;
- **EventFilter:** Second part of the HLT system. Restricts even more the selection done by the LVL2 trigger. Is also used as a monitoring and calibration tool for the detector, as it is able to provide samples of acquired data for quality checks.

The following diagram shows the interaction between the Online Software and other systems:

FIGURE 2.1. Online Software and other Trigger-DAQ systems



In order to better understand the role of the Online Software, it is useful to describe in a very abstract fashion how the data taking process is distributed by the above mentioned components:

- If the LVL1 trigger decides that a piece of data is interesting, this data is retrieved from the RODs and made available by the DataFlow on special buffers;
- The DataFlow then provides the data to the LVL2 trigger for the next round of selection²;
- If the LVL2 decides that there are chances that the data is interesting, the rest of it is read from the RODs into the DataFlow;
- Since the detector (RODs) only provides fragments of events, the DataFlow also provides a reconstruction service that assembles the previously collected event pieces in full events;
- Finally the full events are sent to the EventFilter for final selection. If decided that the event is good, it is sent to mass storage.

All the previously mentioned systems need however to synchronize and communicate in order to cooperate correctly. Another important point is the fact the trigger-DAQ is designed to be a highly configurable system, being that there is also the need for parameterization of the various hardware and software components.

More specifically now, the Online Software is the framework providing specialized control, configuration and monitoring to the sub-systems mentioned before, but, as addressed in the technical proposal for high-level triggers, DAQ and DCS [2], it “*excludes the processing and transportation of physics data*”. Another interesting feature of the Online Software is the ability to divide the data acquisition in partitions. As described in ATLAS DAQ Prototype -1 Technical Note 60 [3], “*par-*

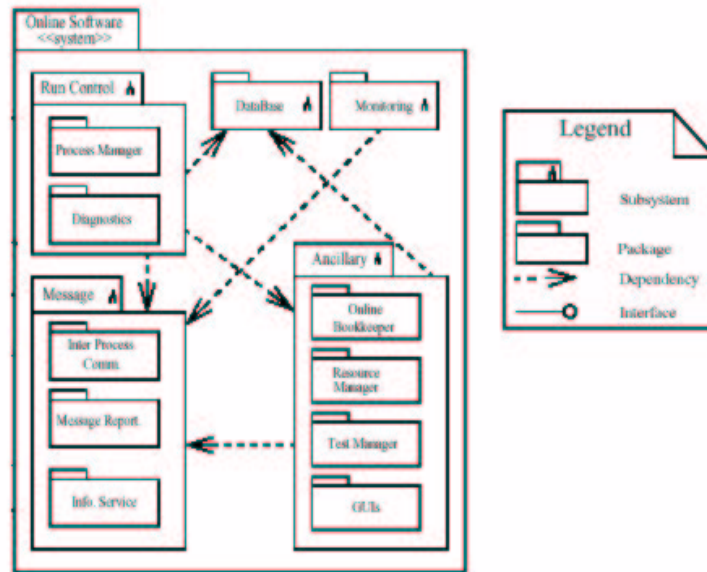
2. It is also possible to do data acquisition for special purposes using only LVL1 trigger, without the High Level Trigger, although this discussion surpasses the objectives of this document.

2.1. High Level Requirements

tioning is defined as the logical and/or physical separation of the experiment into mutually exclusive sub-sets (partitions)”. In other words, the Online Software allows for the separate operation of different hardware subsets of the detector concurrently, by enforcing the exclusivity of it’s use and deploying the necessary software.

The Online Software package itself is divided into component groups (or super-components), as can be seen in figure 2.

FIGURE 2.2. Online Software’s internal structure



Each super-component tackles one of the groups of functionality that the Online Software provides, namely Run³ Control, Messaging, Databases, Monitoring and Ancillary. It is to be noted also that the individual components that form these groups are themselves based on other packages, which are used as software build-

3. A run is a continuous period in time of data taking using a given hardware and software configuration and a defined set of run parameters. It is identified by a unique run number.

ing blocks. These may be homegrown or outsourced. These packages provide functionality such as inter-process communication tools, persistency, graphical APIs, etc.

Addressing now the functionality of the super-components, the **Run Control** is generically responsible for assigning commands to the systems under the Online Software's responsibility. Returning to figure 1, one can see that the Online Software communicates with other systems by means of interfaces, called Local Controllers. These Local Controllers are specialized pieces of software able to translate the commands coming from an hierarchically superior controller into meaningful operations in the particular system's context. This is how the framework is able to control an heterogeneous group of hardware and software components.

The **Messaging** super-component manages all the communication infrastructure made available. This functionality is particularly relevant in the framework of this dissertation, so it will be useful to describe the purpose of some of the participating components:

- As defined in the Online Software's home page [1], the goal of the **MRS** (Message Reporting System) *"is to provide a facility which allows all software components in the ATLAS DAQ system and related processes to report error messages to other components of the distributed DAQ system"*;
- On the other hand, the **IS** (Information System) *"provides a mean of inter-application information exchange in the distributed environment"*.

The concepts of IS and MRS will be refined further ahead in this document.

Continuing the description of the super-components, the **Monitoring** is responsible for physics and detector data sampling and displaying - for quality check and calibration purposes - and the **Ancillary** provides less critical services, such as a graphic control panel for run supervision, diagnostic services, etc.

Finally, the **Databases** super-component is, at the moment of the writing of this document, split into: **Configuration** (ConfigDB) component, which holds persis-

2.1. High Level Requirements

tently and gives access to configuration sets for different data acquisition modes (i.e. parameterization of software and hardware DAQ components); **Online Book-keeper**.

The OBK is described in the technical proposal as the component which “*archives information about the data recorded to permanent storage by the DAQ system. It records the information to be later used during data analysis on a per-run basis and provides a number of interfaces for retrieving and updating the information.*”

It can be seen as the tool that records DAQ events (i.e. DAQ metadata) in a temporally ordered fashion for the Online Software. This data is important not only to maintain the coherency of the Online Software, but also to provide input on data acquisition conditions during physical data analysis time.

2.2. Previous Investigation

The starting point for this dissertation was not zero, as some work had already been developed in the previous three years. Namely, a first OBK prototype using C++ and Objectivity/DB for persistency (OBK/Objy) was built and deployed successfully. The study and enhancement of this prototype in the scope of the current MSc framework was an important step into understanding the problem in general and on how to proceed. The points mentioned in the paragraphs that follow describe what were the premises from which the current work started.

2.2.1. Use of OO

The first prototypes of possible solutions to the Book-keeping problem relied heavily on Object Oriented databases. At the time of the start of the project (1996), CERN's IT/database group was recommending the experiments OO databases for solving the problem of HEP data persistency both at the level of the complexity of the data models and the challenge of storage scaling. Inside the Online Software group this general suggestion was followed and both the components that provide database functionality (the OBK and ConfDB) use OO persistency tools.

In this context it is useful to mention the RD45 project at CERN [4], which was responsible for the investigation and proposal of solutions to the problem of storing and managing HEP data, expected to go to level of hundreds of petabytes when all the LHC experiments are in production.

The basic problem faced by the RD45 project was the fact that traditionally the persistency problem (and a lot of other computing problems) were tackled with homegrown solutions, coded usually in FORTRAN. Although this was the trend for a long time, these solutions are simply not adequate to the scale of modern

2.2. Previous Investigation

HEP. Also, the pure relational data model was reaching its limits in terms of describing modern physics data.

It seems important to mention at this point that the focus of the research of RD45 was on commercial industry standard solutions - at this time already more or less⁴ powerful enough to be applied.

After three years investigation, the recommendations of RD45 could be summed up as follows: OO persistency tools ODMG (Object Data Management Group) compliant are the better option to store HEP data, mainly because of the low impedance between the transient and persistent object models, the capability of replication, schema evolution and object versioning, and in general offer more flexibility and functionality than previous approaches. The possibility of interfacing an OODBMS coupled to a tertiary Mass Storage System (MSS) for high volumes of data makes this solution even more interesting - today's HEP data for some of the most significant experiments (e.g. Atlas, BaBar) are being stored using Objectivity/DB and HPSS or CASTOR as the tertiary repository.

As mentioned before, the Online Software followed these guidelines and adopted Objectivity/DB as it's long-term data management and backup system. Another light-weight persistent object manager called OKS (Object Kernel Support) [5] was selected to address the real-time requirements of the project. This latter persistency tool is an inhouse solution.

2.2.2. *Objectivity/DB experience*

The first implementation of the Book-keeper for the Online Software was developed from 1996 to 2001 and, as explained in the above discussion, used Objectivity/DB. Although the present work was only started in 2001, much from this previous work must be taken into consideration while discussing a sound proposal for a production Book-keeping system. Further ahead in this document technical data on this first OBK prototype will be presented in order to sustain a coherent

4. For instance in BaBar a series of hacks to the kernel of Objectivity/DB were needed. They actually gave a serious boost to Objectivity to improve it's product scalability.

technology comparison with other approaches to the problem. For the time being a brief description of what was already known/supposed on Objectivity/DB use is valuable in understanding the starting point of this work and subsequent options.

Objectivity/DB is a pure object-oriented DBMS, commercial package. As such it provided a good test-bed for learning the technology, as well as a laboratory for testing possible architectural configurations for the OBK. This first prototype was implemented in Lisbon by Prof. Antonio Amorim and Andre Ribeiro.

The introductory work for this dissertation consisted on becoming familiar with the existing software by implementing a web-based retrieval mechanism for the already existing Objectivity/DB databases acquired during test-beam⁵ activity, as well as fixing a number of bugs in the OBK/Objy prototype. These activities went on for a number of months, and some of the acquired knowledge and experience is described in the following (unordered) points:

- The schema definition mechanism (ODMG compliant) is very powerful and reduces the gap between transient and persistent objects. This provides a high degree of flexibility and enables the programmer to set a mind frame on a high level language paradigm - OO - throughout all of the development process;
- The Objectivity/DB databases at ATLAS are served by a reduced number (around 3) of machines, some with MSS connections and running AMS (Advanced Multithreaded Server) for remote requests. The previous book-keeping test-beam activities were stored and accessed using these machines;
- Being a commercial tool, Objectivity/DB provides the user a significative number of powerful features, including graphical browsing utilities, transaction and recovery management. It has however the disadvantage of requiring a licence, which raised problems in an Open Source development environment such as the Online Software;

5. A test-beam consists of a period of time where a part of the detector (sub-detector) is put to test. Clearly to test one must get the output of the detector. The DAQ metadata can be stored in the OBK.

2.2. Previous Investigation

- Objectivity/DB adds another step to the development - it is necessary to define the persistent objects in DDL (Data Definition Language) and pre-compile them to generate the schema: in the OBK case the coding is done in C++, so the generated schema files are C++ that is compiled and linked with the user code.

2.3. Structure of the problem

As already mentioned, the pragmatic goal of the present dissertation is to, in a sustained way, be able to recommend adequate resources for a Book-keeper in the context of ATLAS' Online Software. The practical approach that was used consisted in building iteratively several increasingly sophisticated prototypes in order to study architectural and technological solutions, as well as respond to new or updated old requirements for the system.

As for all problems of a certain degree of complexity, the best way to deal with the work is to split it into several areas of activity. There were certainly several fields that had to be exploited in parallel in order to sustain the research. They can be formalized as follows:

- **Research into the DBMS area of knowledge:** a considerable amount of time was spent understanding the database (VLDB) discipline, also to be able to academically place and evaluate the work;
- **Development and maintenance:** the developed prototypes were actively used as part of the Online Software in tests and test-beams. Clearly, for each prototype was necessary to follow a pre-defined software development process and to maintain it as it was part of the Online Software periodic releases (approximately every 3 months);
- **Evaluation of the work:** a very important part of what was done was to be able to evaluate - without that nothing would be learnt. There are several degrees of evaluation, including functionality, efficiency, architectural and technological worries, for each one of the implemented prototypes.

The next points will try to elaborate a bit more on these three topics.

2.3.1. Research into DBMS

The IT/Database group is quite active at CERN, as can be seen from the already mentioned RD45 project. In fact HEP to some extent pushes the discipline for-

2.3. Structure of the problem

ward. As such, the environment couldn't be better for the kind of work that was done. Also the University of Sunderland provided its library in-site and online (WWW) facilities in order to access books and other publication. CERN's library has also a good publication repository.

The research consisted then mainly of reading relevant books and papers and also in publishing when possible the attained results. Attending conferences constituted very positive steps into acquiring an overview vision of the subject. The articles on the OBK presented at the CHEP [44] (Computing in High Energy Physics) and VLDB [48] (Very Large Databases) conferences can be found in appendixes C and D.

2.3.2. Software developing environment and process

The OBK prototypes were implemented following the software development process and using the environment suggested by the Online Software group. The process is quite straightforward and was very useful in preparing the material so that valid conclusions could be extracted from the work. The basic idea is that every component follows a cascade approach to the development, being possible to go back from a phase to a previous one. The contemplated phases are: **Requirements** - the problem is first stated and structured; **Pre-design Investigation** - technologies possible of being used are evaluated; **High level Design** - abstract view of the system, outlined in a specification language such as UML (Universal Modelling Language); **Detailed design and Implementation** - mapping of the problem onto a programming language; **Unit testing** - checking of the functionality, efficiency, etc. against the requirements; **Integration** - integration and test with other components; **Maintenance**.

The attraction of this approach is that every time that a new component (or prototype for a component) is implemented, the same set of guidelines is followed. This makes it very easy to track, in the case of the OBK, what are the main differences from prototype to prototype, gains or losses in efficiency, elegance, speed, etc. Important documents generated by the process are the URD (User Requirement

Document), design document, implementation note, user's manual, test plan and the test report.

Also it needs to be mentioned that there are a number of tools to support the development that contribute to the automation of the process. For example, CVS is used as the code repository, SRT and CMT (both homegrown solutions) are used for release management, Rational Rose for high level design, Perl and Unix shells for scripting, Insure and Logiscope for code checking and verification, Framemaker for documentation generation, etc. This, added to the fact that several languages are used to code the system itself, makes the software development environment quite heterogeneous and advanced.

2.3.3. Evaluation

In order to present results, a very important part of the work was to be able to evaluate each one of the iterations of the adopted prototyping process. There are many issues of the problem that required analysis, e.g. functionality, efficiency, architecture, use of technology, integration with other components, speed, scalability, etc.

The software process used inside the Online Software already presupposes a test plan document and a test plan report. These two documents can provide a basis for some evaluation and comparison of the results of several iterations. Some of the tests that can be included in this phase are related to functionality vs. requirements, speed, scalability and stress.

Other very important sources of information are the scalability tests and test-beams executed regularly as part of the Online Software schedule of activities. In the scalability tests all of the system is put to work in a controlled environment, for scalability, stress and speed tests. These provide a very useful test-bed, as well as for issues concerning integration with other components. The test-beams are even more important as the software is supposed to respond in a real world situation. Feedback from the users during and after (use of the available data retrieval mech-

2.3. Structure of the problem

anisms) these tests constituted very valuable information. Some users also belong to foreign institutes participating in the experiment and performing their own tests.

Finally the feel and feedback from the publications and attendance of specialized conferences provided a way of matching the work done with similar research. Again to be mentioned is the importance of reading good text books and other publications (papers) as an evaluation and comparison criteria.

2.4. Summary

The OBK is a software component of ATLAS' Online Software, which on its turn is a sub-system of the Trigger-DAQ. The Trigger-DAQ is responsible for acquiring, filtering and moving the physics data from the detector to mass storage, while the Online Software keeps the components of the Trigger-DAQ working together. More precisely, the Online Software provides control, configuration and monitoring services to the Level 1 Trigger (LVL1), the detector, the Dataflow, the detector control system (DCS), the Level 2 Trigger (LVL2) and the EventFilter. It is composed of several components grouped into five super-components: Run Control, Messaging, Monitoring, Ancillary and Databases. The OBK is part of the Databases super-component and is generically defined as the component which *“archives information about the data recorded to permanent storage by the DAQ system”*.

Investigation had been made before work on the present dissertation started: a first OBK prototype using Objectivity/DB for persistency had already been built which allowed a first study on architecture and technology. The results from this work were used to perform a first approach to the problem.

After this previous investigation, the problem was structured into three main activities: research into the database area of knowledge; development and/or maintenance of three OBK prototypes based on different technologies (Objectivity/DB, OKS and MySQL); evaluation of the performed work. One of the main challenges while developing the present MSc was to be able to interconnect these activities in a coherent a productive fashion.

Databases (in High Energy Physics)

In chapter 3 some insight is given in the specific field of knowledge this dissertation is focused on - databases, in particular databases in High Energy Physics.

Section 3.1 (Database Concepts) provides the basic concepts on databases, how the field of knowledge is structured and also gives some insight on the several stages of the technology evolution until now.

Point 3.2 (Databases and HEP) shows how High Energy Physics uses databases to suit it's needs for mass storage of physics data. After an historical introduction, current techniques are explored and some of the problems faced by modern HEP database computing are laid out.

3.1. Database concepts

Database technology emerged in the 1960s, as a response to the growing need of mechanisms for abstracting from storage details and concentrate data management services. This need came about due to the problems encountered by programmers when their applications started to store and/or generate large amounts of data¹. These problems were felt because of the increasing difficulty in the maintenance of data being read/written by different applications, in different formats and from/to different places. Dealing with data update and redundancy was becoming a big problem.

Ever since the appearance of the first examples of database technology, the field of knowledge has evolved rapidly, both theoretically and in terms of products offered by vendors to the final users. Currently, the Database Management System (DBMS) plays a fundamental role in organizations, by providing a central repository for data that can go up to terabytes or even petabytes, in some special cases. From the programming point of view, modern DBMSs offer not only central management of data, but also a significative amount of functionality that is shifted from the application to the database system. Instead of being code centric, today's applications are much more data centric due to all the facilities provided by DBMSs.

3.1.1. Database principles

Although the technology has evolved since the first databases, it is possible to identify some characteristics that depict what is essential in a **database**. A very simple definition (from “Fundamentals of Database Systems” [6]) is the following:

“A database is a collection of related data.”

1. Certainly one has to relativize the meaning of “large amounts of data” taking into consideration the exponential evolution of computer hardware technology since the 1960s.

3.1. Database concepts

Although this statement is sufficient to exclude to some extent what is not a database, it is also necessary to say that:

- A database represents a subset of the real world and to be useful should be up-to-date with the part of the world that it represents;
- A database holds a collection of data with some meaning and is structured according to that meaning.

Databases hold all kinds of data, ranging from simple address books to catalogs of products for a store, or even all the integrated information necessary for the management of a company. Modern databases also store multimedia objects, associating image with text and sound.

It is important to define also what is a **DBMS**. Although the concept coincides to some extent with that of a database, the DBMS is not only the data repository but also the set of software tools that enable users to create, change and use a database. It may include software to create database schemas, APIs for interfacing with user applications, general purpose graphical browsers, statistical tools, etc. Modern DBMS packages include a significative amount of tools that help make the life of the programmer or the DBA² easier.

As will be shown in the next sections, today's database manufacturers make available a large range of features in their DBMS products, for example towards distribution, real-time transaction management, object-orientation, mobility, etc. Despite this, there are some features which are common to all DBMSs, as they are in fact the essential advantages of using this sort of system, as opposed to developing your own database management code. These are:

- A DBMS makes available mechanisms to store to, retrieve from and update the database. As mentioned before, these mechanisms may come in the form of lan-

2. Database Administrator - the person who manages a deployed DBMS system for a community of users.

guage specific APIs, database interaction languages, etc. It also provides a set of utility tools to manage the database;

- A DBMS holds not only real-world data but also the catalog of how that data is mapped in memory, making the database an independent entity and complete on its own. The fact that such metadata exists creates a level of abstraction for the application while interacting with the database. This isolation of the application from the database facilitates changes or the evolution in the data storage structure;
- The DBMS centralizes data and makes available common procedures for reading or writing information. This enables several users/applications to access the same data in a coherent fashion while avoiding redundancy. The DBMS provides³ one or more servers with such features as authentication, concurrency management, remote access, constraint verification or backup/recovery services to client applications that connect to it.

3.1.2. Database technology evolution

Database evolution can roughly be divided into three stages or generations. The following text goes through them, trying to provide an insight on how the area evolved in terms of available technology the design methodology.

As mentioned before, the **first generation** of DBMS was born in the 1960s, as a response to the need for persistency mechanisms more efficient than simple flat files completely dependent on the applications that write/read them. These systems were based on Codasyl network models, where the data is stored in trees or graphs

3. The enumerated features may vary from one DBMS product to another.

of individual data containers. Some representatives of this generation are the Univac DMS 1100 [7], General Electric IDS [8] or IBM IMS [9] systems.

Despite being efficient products, these first generation DBMSs were hardly programmer friendly, since the Codasyl model did not promote a clear distinction between the logical and the physical design of a DB application and was too close to the machine. This lack of clarity led to not very well defined design methodologies. On the other hand, this first generation introduced important innovations such as the independence between application and database or accessing a DBMS via the network.

The **second generation** of DBMS originated in the 1970s. These were based on the relational data model by E. F. Codd [10]. A large number of DBMS packages using this model were produced and still have the major share of the database market (e.g. Oracle, Informix, Sybase, etc.). The relational model is simpler to understand from the human point of view than the Codasyl model - its success partly comes from simplicity. The model describes the world by means of tables which can be related or not. In these tables each column holds several instances of a field and each line holds a register about a person or a thing. The SQL declarative data manipulation language (ANSI and ISO standard) is tightly coupled with the relational model and proved successful because of the simplicity of its use - in contrast with previous procedural data manipulation languages.

The relational model came from mathematical research and a significative amount of investigation effort was put into this kind of databases. Topics such as normalization, indexing or query optimization were (and still are) part of the agendas of computer scientists. The relational DBMS products that were developed offer also a range of flavours in performance (e.g. real-time, parallel databases, etc) or in distribution (distributed databases, mobile databases, etc).

From the design methodology point of view the expressiveness of the relational model along with the ANSI three level database architecture (independent exter-

nal, logical and internal layers) and Chen's E/R model gave rise to a clearer database design methodology, divided in three steps:

- **Conceptual design** is the phase where the designer models the part of the world he/she is trying to describe, using the E/R conceptual language for example. The conceptual design is independent from the database model in which the system is going to be implemented as it is intended to be understood by the final user;
- **Logical design** consists of mapping the conceptual model to the database data model used by the chosen DBMS;
- In the **physical design** the emphasis is put in adapting in the most efficient way the logical model to the physical devices where the system is going to be implemented.

Software tools to help the programmer through the database application development cycle (CASE tools) also appeared in this second generation, providing features such as graphical design interfaces to build the models or forward engineering engines to create application skeletons. Their success was however rather limited, due to the fact that the design languages and processes were not unified and the CASE products provided the programmer their own interpretation of the development cycle.

In the beginning of the 1990s the Third-Generation Database System Manifesto [11] was presented, giving birth to what is called the **third generation** of DBMSs. The basic claim of the manifesto was that, despite the huge success and impact that second generation products had in databases, these systems were too focused on business data processing applications and not flexible enough to address less typical problems. In fact, the document went as far as saying that even business applications were not fully covered by the relational model.

The authors of the manifesto presented three tenets as the basic pillars for third generation DBMSs: third generation DBMSs should provide support for richer

object structures and rules; third generation DBMSs should subsume second generation DBMSs; third generation DBMSs should be open to other subsystems.

From the first of the above tenets, one can deduct that the expressivity of the relational paradigm was not enough anymore, as richer database constructs were needed. The response from vendors was to emphasize the pure Object-Oriented DBMS systems (already existing from the 1980s) as the next “big thing” in the database business. Some representatives of this class of systems are ObjectStore, Objectivity or Versant. The ODMG consortium was also formed to reinforce the position of OODBMS in the market, by standardizing the interface (enhancing portability), language bindings and a query language (OQL) for these products (see [12]). As a result of this effort OODBMSs became a competitor to second generation products, offering an wide range of possibilities based on OO technology to the DB programmer.

The relational products were still however firmly implanted - in fact SQL is and probably will be for years to come the general accepted database manipulation language. The relational DBMS manufacturers recognized however that OODBMSs were satisfying segments of the market - for example non-conventional applications, such as HEP - by providing more flexibility. The counter-attack was to extend the SQL standard to incorporate OO features (SQL3, SQL/MM). The mapping of OO in the relational model was however not a linear task, as it involves the merging of paradigms which are substantially different.

The result of this conflict is that at the time where this dissertation is being written the market offers both pure OODBMSs, as well as hybrid OO/relational products, called Object Relational Database Management Systems (ORDBMS), that resulted from the evolution of previous second generation DBMSs.

From the design methodology point of view the OO approach to databases introduced new ways of doing things, and in fact contributed to the “settling down of the dust”. The two-tier (client/server) and three-tier (interface/middleware/database) reference models clearly defined the roles of the objects involved in the database pattern architectures. Moreover, applying OO to the conceptual/logical/physical design cycle removed barriers between the several phases, speeding up

and easing the development process. The standardization of UML as the recognized software object modeling language was also a boost for programmers confidence in OO database design (either using pure OO or Object Relational DBMSs).

3.1.3. Current picture

The DB landscape currently (2001/2002) is quite diversified and rich, not only in terms of data models, but in various other dimensions. In the following lines some of those directions are superficially described, the ones that are more directly related with the theme of this dissertation.

In terms of performance, modern database systems are taking advantage of hardware evolution in order to allow faster store and query times, using for example the increasingly parallel systems available in the hardware market. Cheaper memory offering greater capacity also contributes to this trend - it is now possible to have 1G of main memory, which only 10 years ago was considered a Very Large Database on it's own. This made possible the appearance of real-time constrained or main-memory databases that alleviate the programmer from the task of coding all the restrictions required by a high performance system of this kind.

In what concerns scalability issues, databases can store enormous amounts of data on disk (secondary storage) and in Mass Storage Systems (MSS). As mentioned before, for instance in the case of HEP these databases can hold up to petabytes of information, which actually pushes DBMS manufacturers to provide systems that scale to these orders of magnitude. The problem is not always trivial to solve, since these applications rely heavily on distributed databases.

It's important to mention also the growing importance of interfacing between the Web and DBMSs. Increasingly the database client applications run on WWW browsers connecting to database servers on remote machines. DBMSs such as MySQL for example have been successful partly because of the ease of interfacing with the Web, due also to scripting languages such as Perl or PHP. In the present

time almost all DBMS vendors recognized the need to include WWW interfaces in their packages.

The work described in this dissertation is outside the market addressed by commercial DBMSs, and can be considered by some authors (e.g. Mario Piattini and Oscar Diaz [13]) a “nontraditional application” (along side others in the field of scientific applications, medical systems, geographical information systems, etc). The problem with current DBMSs while dealing with these kinds of problems is that they are monolithic, offering in one package a whole range of features at a high price, despite the real needs of the users. For this reason HEP has been traditionally developing it’s own database management systems, although the current amounts of data to store already justifies a more professional approach to the problem.

In the next section databases in high energy physics will be discussed, pointing out how this specific branch of database technology interacts with the whole of the discipline and what synergies can be drawn from those interactions.

3.2. *Databases and HEP*

HEP has traditionally been an intensive user of database techniques and products, normally at quite higher scales than normal market demand. Today's databases in HEP is struggling in several directions simultaneously, in terms of: data models - OO, Object-Relational, Relational; using commercial or homegrown solutions; distribution of the massive amounts of data produced by modern HEP experiments to regional data centers or institutes - interfacing DBMSs with the GRID is one of the anticipated solutions.

In this section the area of databases in HEP is discussed, bringing some light to the problem this dissertation addresses. Although the amount of physics metadata stored by the OBK is orders of magnitude under that of physics data eventually stored by ATLAS, much can be learnt from the problems faced in the storage of regular physics data. If one considers the OBK as a scaled down version (in size and complexity) of a physics database, addressing such problems as what DBMS to use, scalability (interfacing with mass storage systems) or performance (being able to respond to the soft real-time requirements of the full Online Software) is simpler, as they are a subset of larger problems tackled by the ATLAS collaboration or the full HEP community.

3.2.1. *Some history*

The history of databases in HEP is largely dominated by in-house solutions, mainly in FORTRAN (extended by the ZEBRA library) - the preferred software development language for the earlier generation of physicists doing computing. At that time (before the 1980s) the regular database market was simply not mature enough to address the needs of HEP experiments, which meant that specialized solutions were absolutely necessary.

Later, relational DBMSs started to make their appearance in the HEP world, although mainly to handle book-keeping metadata and not the raw or reconstructed⁴ physics data, which was still handled by specialized solutions. It was also during the 1980s that the HEP community started to shift its attention to the OO

programming paradigm, which permeated to the databases layer of physics' software. To be said also that the employment of software engineers in HEP experiments also gave a boost to the quality of produced software and linked the HEP computing world to the "pure" computing discipline.

In the 1990s the RD45 [14] project at CERN took as one of the premises for investigation on databases for future HEP experiments the use of commercial DBMS packages. At this time the HEP community was increasingly interested in the OO paradigm, as it is a closer match for physics' needs in terms of data model than previous relational solutions. The standardization of the OO database model by ODMG also added confidence to that choice. As mentioned in the previous chapter Objectivity/DB was the selected OODBMS by the LHC experiments at CERN and also in other experiments, such as BaBar, AMS, etc.

In 2001/2002, the certainty of the choice of a OO commercial DBMS package is again being questioned, as other players also present strong arguments as the candidates for solutions to physics' data storage. As was mentioned in the previous section, commercial Object-Relational databases propose a hybrid data model, coupled to tens of years of experience in databases by enterprises such as Oracle, etc. Another option can be found within the HEP community, as ROOT⁵ developers also present solutions to the problem.

3.2.2. *Current trends*

Objectivity/DB is, at the time of the writing of this dissertation, the official DBMS for LHC long term data. As mentioned before, the first prototype for the OBK was implemented using that same DBMS. It is well known however that the software world changes at a very fast pace and so does the databases in HEP.

-
4. Physics' raw (detector collected) data is passed through reconstructing algorithms to facilitate offline analysis.
 5. Previously PAW, is the most successful framework for physics' data analysis. The framework also encompasses tools for object persistency.

Given the very large sizes of modern HEP databases, computer scientists working in the area are increasingly concerned about how to physically store all this data and how to distribute it afterwards to the scientists in local institutes, geographically dispersed around the globe. In what concerns mass storage systems the solution is normally based on tapes, managed by some software system such as HPSS (High Performance Storage System). CERN has developed its own MSS system called CASTOR (CERN Advanced STORAge system). Other interesting solutions can be found for instance in Jefferson Lab with JASMine [15] (based on JAVA) or Fermilab with SRM (Storage Resource Managers). Both these experiments also interface their databases with GRID tools [16] for distributing data to interested parties. This functionality will be increasingly important, as a significative investment is being done on GRID technology and setting up the various geographically dispersed tiers of database by which the physics information will flow is one of the worries of today's HEP computer scientists.

In terms of persistency solutions, at the time of the writing of this dissertation the "market" of databases considered by HEP offers quite a few options apart from Objectivity/DB. In the context of CERN the LHC experiments are trying several solutions to see which one better suits their needs. These solutions are:

- **Object/Relational:** Always considered before as a tool for management of organizational or commercial data, Oracle presents with its 9i version a set of features very attractive for databases in HEP. In [17] the CERN database group provides a preliminary analysis of this sort of tool, explaining that Oracle claims to have implemented in this version SQL:1999 (a rich OO data model merged with the relational one). It provides also C++ binding, as well as strong VLDB features that look promising in terms of storage scalability;
- **Pure relational:** Although not powerful or expressive enough to handle physics' data, the state of maturity of pure relational DBMSs is very interesting. Systems such as, for instance, MySQL, offer a price/power ratio which places them among the most used DBMSs in the world today. HEP also employs this technology very effectively in smaller metadata databases for example for control or book-keeping purposes;

- **ROOT I/O:** The developers of the former PAW analysis framework evolved this product to the ROOT system, which is object oriented and more functional than the previous package. The framework grew also to include general purpose language's functionality such as distribution or object persistency. Despite the fact that ROOT is not a DBMS, the persistency mechanisms that it provides are quite attractive to the HEP community, as they result from the group's long experience in computing in HEP and are tuned to match physics' data storage, retrieval and analysis requirements. In [18] ROOT developers explain the reasons why they claim that ROOT I/O should be preferred over a general purpose OO database for physics' data management.

3.2.3. *Current problems*

The problems of databases in HEP amplify those of databases in general, as HEP in fact pushes the existing technology to the limits⁶. An essential difficulty of current DBMSs while addressing HEP (or other) problems is the fact that software database packages are offered as full blocks of functionality, hardly modularized. This bulk of functionality is of course a trade off with efficiency, by which reason some experiments are increasingly turning their attention back to in-house HEP tuned solutions, such as ROOT I/O.

In [19] the fundamental needs and wants of the HEP community in what concerns databases (a DBMS system) are described. The following points enumerate only a subset of them, which is closer to the scope of this dissertation:

- Address large amounts of write-once-read-many data. For physics data the quantity can go up to tens of petabytes. The data stored by the OBK over the lifetime of the ATLAS experiment should be one to two orders of magnitude less;
- Support addition of significative amounts of data on a daily basis while keeping the system still accessible for querying;

6. An example is the involvement of the HEP community in defining the ODMG standard.

- Support simultaneous queries;
- Support data access over an international wide-area network;
- Provide a flexible data model which supports versioning and schema evolution;
- Interface adequately with tertiary storage, providing support for storing and querying the databases efficiently.

While considering the OBK, another aspect of the problem is how to provide the physicists doing analysis an integration between the physics data and information stored in the OBK (since the OBK holds the conditions under which the physics data was acquired). Some sort of integration (in principle not physical) of the physics data with the OBK's data will become necessary.

3.3. *Summary*

A database can be generically defined as “*a collection of related data*”. One can restrict the definition by saying also that: a database represents a subset of the real world and to be useful should be up-to-date with the part of the world it represents; a database holds a collection of data with some meaning and is structured according to that meaning. The concept of DBMS can be built on top of the one of database: a DBMS makes available mechanisms to store to, retrieve from and update the database; a DBMS holds not only real-world data but also the catalog of how that data is mapped in memory; a DBMS centralizes data and makes available common procedures for reading or writing information.

Database evolution can be fitted into three generations. The first one was born in the 1960s with systems based on Codasyl network models. The second started in the 1970s with products based on the relational model by E. F. Codd. Up until today relational systems still hold by large the largest share of the market. This trend can be attributed to its ease of use, coupled with the success of the SQL query language and the clear design methodology associated with the model. Finally, the third database generation started in the 1990s with the Third-Generation Database System Manifesto. Third generation database philosophy emphasizes rich object structures and rules, reason why Object-Oriented database DBMSs boomed in the beginning of the last decade. Despite the failure by OO databases to reach general acceptance, OO features are currently offered with all the best known relational products (e.g. Oracle).

In what concerns the HEP world, the history of databases is highly dominated by in-house solutions. This trend is due to the specificity of the problems posed - high volumes of data, complexity of the data structure - which is not easily tackled by conventional DBMS systems. However, the standardization of the expressive OO model for databases interested the physics community, which in the 1990's adopted Objectivity/DB for LHC experiments. In our days opinions split again, as Objectivity/DB proved not being the ideal tool for the job. Other options such as Object/Relational Oracle (9i) or ROOT I/O (homegrown solution) seem to be also

valid candidates. MySQL seems to be also a cheap and efficient alternative, especially in what concerns secondary databases for management of auxiliary data.

*Physics' metadata
gathering for ATLAS'
Online Software*

Chapter 4 is intended to be a study of the OBK from a conceptual point of view.

In 4.1 the components of the Online Software that the OBK interacts with are identified and examined in order to provide a basis for a conceptual design and later for the implementation.

4.2. aims at establishing a conceptual design of the envisaged book-keeper system. This is done in two phases: in the first the high-level requirements are laid out; in the second a conceptual view of the system is presented.

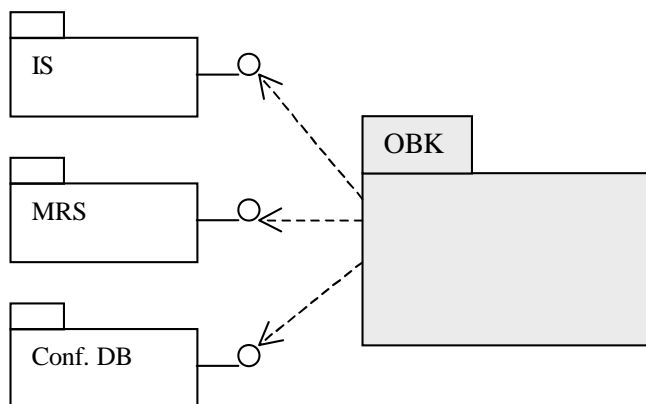
4.1. The Online Software from the OBK's viewpoint

After putting the problem to address into perspective (chapter 2) and evaluating the technology available (chapter 3), the next step is to devise a conceptual design of the system. This conceptual design will provide an abstract starting point for specific technology dependent implementations of the OBK.

In order to arrive to this conceptual design it is necessary to analyze not only what are the requirements for the system, but also in what fashion the OBK will interact with other Online Software's components and how these components behave.

In 2.1 (High Level Requirements) the main goal of the OBK is already stated generically as being the archival of information “*about the data recorded to permanent storage by the DAQ system*” [2]. In [20] a first refinement of these high level requirements defines as main data sources for the OBK the Online Software's messaging mechanisms (MRS and IS) and the Configuration Database components (figure 4.1). In the next section these Online Software's components/areas are studied and understood from the OBK's viewpoint.

FIGURE 4.1. Dependencies between the OBK and the Online Software



4.1.1. Communication infrastructure

The Online Software uses two high-level messaging mechanisms for communication: the MRS and the IS. Before entering a description of these packages it is useful to understand the underlying communication infrastructure CORBA on which they are based. A short discussion follows:

An OMG CORBA [21] compliant system called ILU (Xerox's Inter Language Unification) was evaluated in [22] as a possible basis to implement a transparent means of communication for objects, independent of their location. In other words, a mechanism that would enable message passing between objects located in the same process or in different processes (running in the same or in different processors) was thought necessary.

ILU implements a client/server communication mechanism which enables the calling of methods of server objects by clients running anywhere. Apart from the fact that clients and servers can exist on different computers, they can also be implemented in different languages since ILU provides several language bindings.

The ILU package was however considered too low-level for the Online Software's communication needs. The IPC (Inter Process Communication) package [23,24] was built on top of ILU to provide the needed abstraction. Besides avoiding dependencies on a specific CORBA implementation, IPC also supports the logical notion of DAQ partitions. Since several partitions may be run in parallel, multiple instances of the same software objects may have to co-exist in the same namespace. The IPC provides for the partitioning of the namespace in order to avoid conflicts.

The full IPC mechanism at run-time consists of one process running a default IPC server (for partition name management) and zero or more processes running named IPC servers (for object name management inside a partition). When a client wants to contact a remote object registered in some partition it may do so by: contacting the default IPC server in order to resolve the named partition (finding the partition's namespace server); searching the object's name in that partition's namespace IPC server; retrieving the object's handle in order to be able to execute

it's methods. The details of resolving the name of the object into a handle (communicating with the IPC partition namespace servers) are mostly hidden by the IPC classes' interfaces.

Although the above IPC overview is quite brief, a detailed study of the Online Software's IPC functionality will not be performed here since it is out of the scope of this document. In the next section the IS and MRS messaging mechanisms will be described. Both of these components rely on the IPC package for communication purposes.

4.1.1.1. MRS

The Online Software's message reporting system's goal is to provide a uniform error and information messaging mechanism for all the ATLAS DAQ. Briefly, the MRS allows for the connection of senders and receivers to an MRS server process. This server distributes messages published by the senders to all the receivers that may be interested in them. The fact that the message routing, filtering and assembling is centralized avoids putting too much processing weight on either the sender or the receiver side of this kind of service. MRS enforces standard well know message types across all the DAQ. All the message types available are stored in a database maintained by the MRS server. There is also the possibility of editing the available message types in the server.

In [25] the full requirement list for the MRS component is presented. Since the system is supposed to serve the full ATLAS DAQ, the requirements list is long and describes topics such as message reporting, message transport, message reception, message filtering and message logging/browsing. Fault tolerance is also contemplated as an important issue - for instance the OBK relies heavily on the MRS system for correct operation.

As was mentioned before, the basis for all the communication within the Online Software is the IPC package. In a way the MRS creates still another more abstract messaging layer, since the system's API is to be used by all of the DAQ and should be as simple as possible. The system's operation at run time can be roughly

described as follows: a sender process connects to an MRS server (one per partition) in order to be able to pass out messages to other components. The MRS server redirects the handling of incoming informations to one of the multiple private MRS servers. When a client process connects to the MRS server and requests certain types of messages¹, all² the messages of the requested types produced by senders that fulfill a subscription criteria are passed to that client through the dedicated private MRS server.

There are three MRS APIs made available to the users: the sender, receiver and command API. The OBK acts as an MRS receiver³, which means that an object of class *MRS_receiver* has to be instantiated and the *subscribe* method has to be called with a subscription expression and callback function as arguments. The callback function is a method in the user's code that will be used by the MRS mechanism to pass messages to the receiver.

It was already mentioned before that MRS messages comply to types as defined in an MRS database. These types adhere to a structure which is formed by: a message ID; message text; parameters; qualifiers. This flexible container allows the different parts of the DAQ to define and contextualize the MRS message types according to their own environment. It is important for the OBK to have knowledge about the messages' data structure, since the correct book-keeping data storage will

-
1. The subscription is formulated by passing a regular expression to the server providing the range of messages in which the client is interested.
 2. In fact the MRS implements the concept of filters, although this functionality is beyond the scope of the OBK's interaction with MRS.
 3. Up to the moment when this dissertation is being written the OBK has only used the receiver services of MRS. In the future the OBK may also become an MRS sender.

depend partly on specific MRS messages or on MRS messages' specific parameters as will be explained later in this document.

4.1.1.2. IS

The Information System's goal is to make available information about the run-time internal state of DAQ software components to other DAQ software components. The IS is different from the MRS in that MRS only provides distribution of messages that are notifications of some event. The IS makes available data values on a continuous basis, which means that, while an IS data value is published by a source, all the receivers that subscribed to that data value can access it at any time. This is a substantial change from MRS, where the messages are instantaneous and the receiver must subscribe in advance in order to be able to get them when they are emitted.

The requirements for the IS are described in [26]. According to them the system should, in general terms, be able to provide a means for sender applications to publish their internal states and update them at regular time intervals, when the data in question changes or on an explicit request from a receiver. The receivers should be able to have continuous access to an updated version of data published in the IS, individually or via logical groups.

The IS also supports the notion of a DAQ partition given the fact that an IS server that supports the dynamic database for published IS objects is always started in a given partition namespace. Unlike MRS it is possible to start several IS servers in the same partition namespace - this makes possible the separation of IS published information into logical groups.

About the user's interfaces for IS, as for MRS there exists a sender and a receiver API. The OBK will be an IS receiver, since part of the book-keeping activity will involve storing selected IS messages. Similarly to the MRS subscription, a *ISInfoReceiver* has to be instantiated and the subscribe method has to be called passing

the IS server name, a subscription expression and the callback function. The server name is necessary since there may be multiple IS servers per partition.

When the callback function is invoked on the receiver an IS message containing an update of data the receiver subscribed to is passed. This information holds the following fields: domain name; information name; type (of the IS class, as defined in a type database); value (for all the fields of the IS class); last update time.

4.1.2. Configuration Databases

One of the major requirements for the DAQ's Online Software is that it should be as flexible as possible, in order to accommodate different data taking modes with varied architectural configurations of hardware, software or running modes. To achieve this flexibility it was necessary to enable the system with a parameter database.

In the design document for the configuration databases [27] the requirements are laid out and can be summarized as follows: the configuration databases should have the underlying behavior of a full DBMS, providing such services⁴ as storage of multiple versions of the contents, import/export facilities, schema evolution facilities, querying schemas or data, system management, locking mechanisms and access control. In terms of interfacing with the user the document states that there should exist a way of browsing and editing the contents of a database visually.

In the implementation of the configuration databases, one can distinguish two layers of abstraction. The pure database layer is managed by the in-memory persistent object manager OKS [5]. In fact, the specificity of the requirements of the configuration databases made necessary the implementation of an inhouse solution for the

4. At the time when this dissertation is being written a major subset of these requirements is already implemented.

problem. The OKS was created in order to be a fast and modular in-memory real-time memory database.

The OKS graphical and programming interface of OKS does not however provide an abstraction layer for the configuration databases' users, who do not want to know about specific details of how the persistence is managed but only how they can create a database and access their specific configuration objects. In this perspective the DALs [28] were created to provide the needed layer of abstraction.

The purpose of the DALs is mainly to provide independence from the underlying persistent object manager (POM) and C++ API wrappers for accessing configuration objects. There are several DALs in the DAQ, as the several subsystems found it necessary to wrap the functionality they required out of their specific configuration databases.

4.2. The OBK as an Online Software component

Given that the TDAQ is a long research project, it is hard to define from the beginning all the requirements for each individual subsystem⁵. Each of the subsystems evolved considerably since the project started as DAQ/EF-1 prototype. Not only the subsystems impact on each other as they grow, but also they are influenced by other ATLAS systems from outside TDAQ (e.g. detectors).

Being a component of a TDAQ subsystem (the Online Software), the requirements posed on the OBK have also evolved considerably since the birth of the project. In section 2.1 of this dissertation a very high level description of the OBK extracted from [2] is given, reflecting the initial expectations for the system. The first formal gathering of requirements for the OBK took place in 1997 and the resulting document [29] was the basis for the first OBK prototype implementation.

From 1997 until today, the Online Software (at the time DAQ/EF-1 Backend) evolved in terms of design, technology, performance and scalability. To cope adequately with these changes, a formal review of the system's requirements took place in the beginning of 2002. This review included also the reformulation of the OBK's requirements and made it possible to incorporate functionality requests and new use cases that came up during the several OBK prototype iterations. As was mentioned in chapter 2, an (informal) requirements evaluation was done for each of the implemented OBK prototypes. The OBK requirements document produced in the 2002 Online Software review [30] gathers and concentrates these previous requirements evaluations.

Although the studies contained in this dissertation aim at recommending the technology and design for a production book-keeper system, it is envisaged that until the OBK enters a production phase⁶ the requirements for the system will still

5. Subsystem is a generic term, meaning that the system is part of a larger system in TDAQ's hierarchical structure. The OBK can be considered as belonging to the bottom layer (components), as it can be implemented by an individual or a small team.

6. Officially 4 years starting from the time that this dissertation is being written.

change. At the present time the full ATLAS experiment's integration phase is just starting. It is this document's author opinion that this integration will reveal important problems and needs for the OBK system.

4.2.1. Requirements

The Online Software's 2002 requirements review was an effort to (re)gain a wide perspective of what other ATLAS systems expect from a (close to) production implementation of the Online Software system. The approach to accomplishing this involved some formalism - a requirements document template was used to register the several aspects of what is demanded from the Online Software in general and from each one of it's components in particular. The template is divided into Use Cases, Constraints, Assumptions/Dependencies, Functional Requirements and Non Functional Requirements.

Since the reference to the OBK requirements document has already been given in the previous paragraph, the next points will just highlight the main guidelines for developing the OBK. Functional and Non-Functional requirements are not included since these are derived from the user input, which can be summarized in the Use Cases, Constraints and Assumptions/Dependencies.

4.2.1.1. Use Cases

Data Acquisition: After being started with the Online Software by a TDAQ expert⁷, the OBK will proceed to the acquisition of the specified DAQ data in an automatic fashion without human intervention. The information will be stored on a per-run basis;

Information Updating: The OBK will allow a TDAQ operator⁸ to add comments to runs for the data corresponding to his/her shift period. It will also be possible to

7. The TDAQ expert is responsible for running and maintaining the TDAQ itself. He is responsible for the initialization and the shutdown of the global system or parts of it.

8. The TDAQ operator is responsible for using the TDAQ system to take data during a particular data taking session (i.e. shift).

update the stored book-keeping data as offline analysis or reconstruction processes may require it;

Data Access: Access to OBK's database will be provided for several kinds of clients. Queries will be answered with data structures understandable by those clients (web browsers, the Online Software's IGUI, C++ applications). Frameworks such as ATHENA or ROOT may also act as high level clients of the OBK;

Database Administration: Users with adequate privileges will be able to perform administrative tasks on OBK's database.

4.2.1.2. Assumptions and Dependencies

Dependency from MRS: The OBK will require information from the MRS system in order to know when a DAQ run starts and stops;

Dependency from IS: The OBK will require information from the IS system in order to gather run parameters;

Dependency from ConfDB (Configuration Databases): It will be necessary to access the configuration databases in order to book-keep the DAQ and detector's configurations for each data taking session;

Production: The OBK will record and provide access to information from many data taking sessions along the lifetime of the ATLAS experiment.

4.2.1.3. Constraints

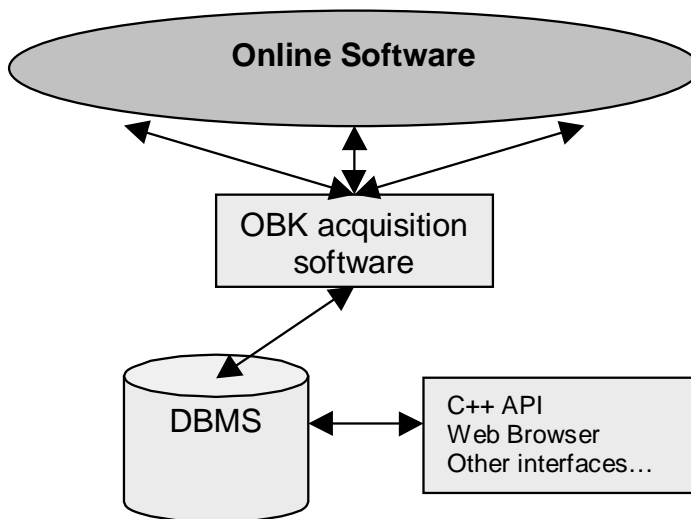
The OBK database shall be accessible during and outside data taking sessions;

The OBK should not interfere with the operation of other components of the Online Software. In case it does, that interference should be measured and minimized.

4.2.2. Conceptual view of the OBK system

From the requirements stated above a simple generic architecture for the OBK can be devised. Figure 4.2 depicts this architecture which is divided into two functionality groups (inside the rectangles): the data acquisition engine, working online with the rest of the Online Software; the database browsing and administration software which works offline from the Online Software.

FIGURE 4.2. Generic OBK architecture



4.3. Summary

In order to devise a conceptual design for the OBK, it became necessary to gather requirements for the system. A first part of this work was to study the Online Software environment and to understand how the OBK could fit in. In order to do that, the several Online Software components from which the OBK gathers log information were analyzed: IS (Information System), MRS (Message Reporting System) and ConfDB (Configuration Databases). Both the MRS and the IS belong to the messaging super-component and rely on the IPC package (a CORBA implementation) for inter-process communication. The difference between the MRS and the IS is that while the MRS's goal is to provide an error and information messaging mechanism for the TDAQ, IS's objective is to make available information about the internal state of TDAQ's components (using a publication mechanism). On the other side, the ConfDB provides a way of parameterizing the TDAQ. Starting a run implies that all the TDAQ software and hardware components will read their configuration from the configuration databases. IS, MRS and ConfDB provide APIs, which means the OBK can easily connect to them to retrieve interesting information.

The next step was to clearly and formally define the user requirements for the OBK software. Being such a long term project, it is obvious that the requirements for the whole of the Online Software change and are refined as time evolves. Despite the fact that a set of requirements had been elaborated for the OBK at the beginning of the project, the most recent revision dates from 2002 when all the Online Software components were reviewed. The requirements document which was produced defines the Use Cases, Assumptions and Dependencies, Constraints, Functional requirements and Non-Functional requirements for the OBK. From all this information it was possible to clearly define a conceptual view of the OBK system.

The present chapter introduces the OBK software developed in the context of this dissertation. It is a full description of the implemented tool, both from the user's and the developer's viewpoints.

5.1. presents the OBK in terms of data acquisition, data browsing and other utilities included in the package. The functionality described is (loosely) common to the three implemented prototypes.

In 5.2. the technical aspects of the OBK are explored and detailed. The text addresses used tools and supported platforms, as well as specific techniques applied in solving each of the available prototypes.

5.1. *The OBK from the user's viewpoint*

In the points that follow an extensive description of the functionality the OBK provides to the user is given. The description is generic enough to cover all the prototypes, although as was mentioned in chapter 4 new requirements were uncovered from the beginning of the project until the moment when this dissertation is being written. If the functionality which is being addressed varies considerably from prototype to prototype, that fact will be indicated in the text.

The information present in the present section can be found in a complete form in the “User’s Guide of the Online Book-keeper for the Atlas DAQ Online Software” [31] (appendix F).

5.1.1. *Data Acquisition*

As was defined in 4.2.2., the OBK can be conceptually divided in the data acquisition module and the data browsing module. The next paragraphs will describe the acquisition module: first the database logical organization is laid out (an issue which is also important to understand data browsing) and then the operation of the OBK is explained.

5.1.1.1. *Logical structure of the OBK database*

From what was exposed in the previous chapters, a natural organization for the OBK database can be the following:

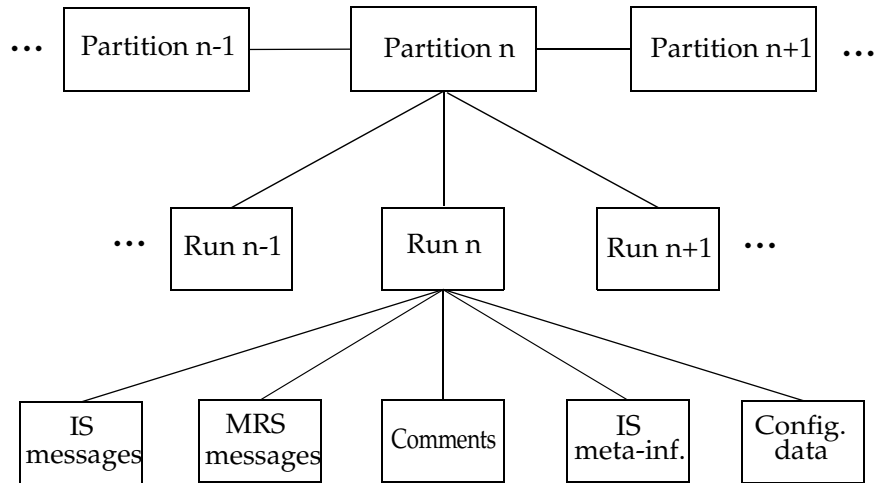
The database is divided into three layers: *partitions* contain *runs* and runs contain *messages* (IS/MRS), *comments* to runs made by the shift operators, configuration data for the run, or *IS meta-information*¹. This way of organizing the book-keeping data respects both the TDAQ *partition* and *run* concepts.

1. The IS defines classes, which are the types of the published objects. The OBK stores on a per run basis all the IS types (if available) corresponding to the IS messages archived in that run.

5.1. The OBK from the user's viewpoint

As will be explained later, physically the database varies from prototype to prototype. However, the logical structure depicted in 5.1. is kept as a reference across the several OBK implementations.

FIGURE 5.1. Logical structure of the OBK database



Runs may be of two different kinds: runs with events (Runs) or slow control runs (SLCRuns). This distinction was necessary because, in a very generic way, the Online Software can be in two states: running or stopped. When it is running, it means that data is being acquired into a Run container in the database. On the other side, when the Online Software is up but in a stopped state (there is no run), the TDAQ components still interchange MRS or IS messages. Although they are less important, the OBK records the messages exchanged during the stopped state in SLCRun containers. The way the transitions occur between Runs and SLCRuns is depicted in table 5.1..

TABLE 5.1. Runs and Slow Control Runs

	Starts	Ends
Run	- MRS Start of Run message is received.	- MRS End of Run message is received.
SLCRun	- a Run ends; - acquisition starts.	- a Run starts.

The OBK data will be stored to the database in the following fashion: a Run is followed by one or more SLCRuns, which in turn are followed by another Run and so on. An example of this sequence is shown in figure 5.2.

FIGURE 5.2. Logical view on a possible run sequence

Run 1000	SLCRun 1000.1	SLCRun 1000.2	Run 1001	SLCRun 1001.1	Run 1002
-------------	------------------	------------------	-------------	------------------	-------------

5.1.1.2. *Operating system environment*

As will be mentioned in 5.2.5, the platforms the OBK runs on are either UNIX or Linux. While using these operating systems, an easy way to parameterize applications is to read values from environment variables. The several OBK prototypes use different environment variables to define their run time behavior. Tables 5.2, 5.3 and 5.4 display these variables and their meanings:

TABLE 5.2. Environment variables for the OBK/OBJY

OO_FD_BOOT	OO_DB_HOST	OO_DB_PATH	OBK_LOG_PATH
Where to find the boot file necessary to load the Objectivity federation ¹ .	Name of the machine where the databases themselves are located.	Path to the directory where the databases are in the host machine.	Path to the directory where the OBK log files are to be stored.

1. An Objectivity federation logically contains one or more databases and their corresponding data models.

5.1. The OBK from the user's viewpoint

TABLE 5.3. Environment variables for the OBK/OKS

OBK_SCHEMA_PATH	OBK_FEDERATION_PATH
Directory where to find the file describing the data model for the OBK database.	Path to the root of the directory structure where the OBK data is stored.

TABLE 5.4. Environment variables for the OBK/MySQL

OBK_MYSQL_HOSTNAME	OBK_MYSQL_USERNAME	OBK_MYSQL_PASSWD	OBK_MYSQL_DATABASE
Name of machine running the MySQL server engine.	User name for the OBK database.	Password for the OBK database.	Name of the OBK database.

Given that the OBK package includes the three implemented prototypes, it is necessary to provide a mechanism to select a particular implementation to be used in conjunction with the rest of the DAQ system. This can be done in the context of the Online Software by setting the environment variable `OBK_DB` to one of the following three values: `OBJECTIVITY`, `OKS` or `MYSQL`.

5.1.1.3. Starting and stopping the acquisition

The OBK is started by default with the rest of the Online Software, unless the *no_obk* option is given while starting the *play_daq*² script. At least an IPC server (for providing the communication infrastructure) and an MRS server must be running in order for the *obk_daq_oks* application to start (although while running *play_daq* this is transparent to the user). Despite the fact that the OBK can also subscribe to IS servers, it is not strictly necessary that one is running for the acquisition to start.

Although in principle the OBK should be started with *play_daq*, it is also possible to run it in standalone mode. To do this it is necessary to set up the proper environment and to run at least an IPC and an MRS server on the partition the

2. *play_daq* is a bash (Bourne Again SHell) script used to start all the Online Software in a synchronized fashion.

OBK is going to be started on. An example (using the OKS implementation) can be found below:

EXAMPLE 5.1.

```
bash$ export OBK_SCHEMA_PATH=/afs/cern.ch/atlas/project/tdaq/
public/
0.0.15/com/obk
bash$ export OBK_FEDERATION_PATH=/home-users/lucio/temp/obk_data

bash$ ipc_server -p my_partition &
[1] 2386
bash$ "my_partition" partition server started

bash$ mrs_server -p my_partition &
[2] 2387
bash$ Loading 2 classes from file "/afs/cern.ch/atlas/project/
tdaq/dist/0.0.14/installed/share/mrs/data/mrs_db.schema.xml"...
Reading 47 objects from data file "/afs/cern.ch/atlas/project/
tdaq/dist/0.0.14/installed/com/mrs/data/mrs_db.data.xml" in
extended format...

bash$ obk_daq_oks -p my_partition
[obk] OBK_FEDERATION_PATH has value: /home-users/lucio/temp/
obk_data
[obk] OBK_SCHEMA_PATH has value: /afs/cern.ch/atlas/project/tdaq/
public/0.0.15/com/obk
[obk] Opening database for partition: my_partition
[obk] subscribe MRS server SUCCESS
[obk] Information: Ready to data aquisition!
```

In example 5.1. the OBK is started in the simplest possible fashion. The command line *obk_daq_oks³ -p my_partition* starts the data collection for all the messages coming from the MRS server that is running for partition *my_partition*;

As mentioned previously, the two types of information servers the OBK can subscribe to are MRS and IS. There is one MRS server running per partition, so the

3. The names of the binaries are different for each OBK implementation. the *obk_daq_oks* binary is called *obk_daq* and *obk_daq_my* for the OBK/OBJY and OBK/MySQL respectively. The list of all the binaries for each implementation and their respective synopsis can be found in appendix A.

5.1. The OBK from the user's viewpoint

subscription only indicates what types of messages one wants to receive from this server. On the other side, there may be zero or many IS servers, which require one subscription per server. Examples follow:

EXAMPLE 5.2.

```
bash$ obk_daq_oks -p my_partition -M "RC_START|RC_END"
```

EXAMPLE 5.3.

```
bash$ obk_daq_oks -p my_partition -n my_is_server -I ".*"
```

In example 5.2 the OBK will store all the RC_START and RC_END⁴ messages that are sent out by the MRS server running on partition *my_partition*.

In example 5.3 the OBK will subscribe to all the MRS messages coming from the *my_partition* MRS server and also to all the IS messages coming from *my_is_server*. The definition of what messages are expected is done by the regular expression ".*". For each IS server one must specify its name (-n switch) and the regular expression (-I switch).

To stop the OBK data acquisition the user should send the OBK process either the INT(errupt) or TERM(inate) UNIX signals. While in the OBK/MySQL both signals produce the same result, in the OBK/Objy and OBK/OKS the TERM signal provokes a rollback on the database. This means that the all the data being stored for the current run is discarded.

Example 5.4 shows a normal interruption of an OBK/OKS process:

4. The OBK must always subscribe to both these MRS message types since the correct run creation depends on them.

EXAMPLE 5.4.

```
bash$ obk_daq_oks -p be_test_single_host &
[4] 3060
bash$ [obk] OBK_FEDERATION_PATH has value: /home-users/lucio/temp/
obk_data
[obk] OBK_SCHEMA_PATH has value: /home-users/lucio/temp
[obk] Opening database for partition: be_test_single_host
[obk] subscribe MRS server SUCCESS
[obk] Information: Ready to data aquisition!

bash$ kill -2 3060
bash$ [obk] Interrupting...
[obk] Exiting normally...
[obk] Exiting normally...
[obk] Exiting...

[4]+  Done                                obk_daq_oks -p be_test_single_host
```

The user may also decide to send a KILL signal to the OBK process. In this case the program will finish without any recovery actions, which means the database may go into an inconsistent state - the most likely problem to occur is that the partition where the data is being acquired remains locked, and no more data acquisition is available for that partition. In this case, the only option is to manually unlock the partition by using special software for that effect (see 5.1.3.).

If ran with *play_daq*, the OBK process is stopped by the script itself which in a normal exit situation sends an INT signal to the process.

5.1.1.4. Annotating book-kept data

An important task of the OBK is to allow shift operators to annotate book-kept data. Annotations consist of human written texts which are appended to a run while it takes place or afterwards. The purpose of this mechanism is to provide extra input to physicists doing physics data analysis.

The several OBK prototypes allow the annotation of book-keeping data in different ways: while the OBK/Objy and the OBK/OKS implement command line

5.1. The OBK from the user's viewpoint

applications, the OBK/MySQL provides a friendlier graphical user interface for that purpose.

EXAMPLE 5.5.

```
bash$ obk_offline_comment_oks -p be_test_single_host
[obk_offline_comment_oks] partition = be_test_single_host
[obk_offline_comment_oks] Enter Run number: 1000
[obk_offline_comment_oks] Enter Run Sub number (hit <return> for
no sub number):
[obk_offline_comment_oks] Enter the repetition number of the run
(hit <return> for the default -0- value):
[obk_offline_comment_oks] Please write the comment (max 300
characters): ola
[obk_offline_comment_oks] Done.
```

Example 5.4 depicts a typical usage of the *obk_offline_comment_oks* command line utility (OBK/OKS). This utility allows adding annotations while the OBK system is not running. If the system is in the running state, the *obk_online_comment_oks* utility will annotate the run which is currently taking place.

Adding annotations in the OBK/MySQL prototype is done using the OBK web browser which is also used to visualize book-kept data (see 5.1.2.).

FIGURE 5.3. Selecting a run to add a comment to

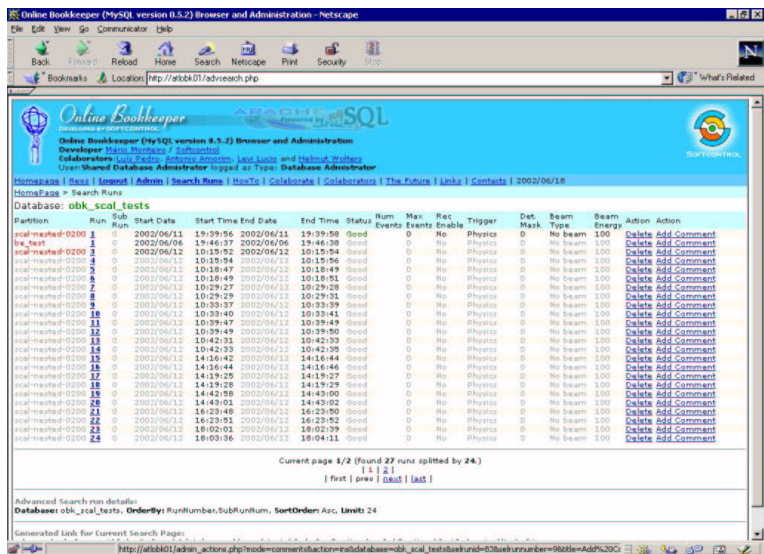
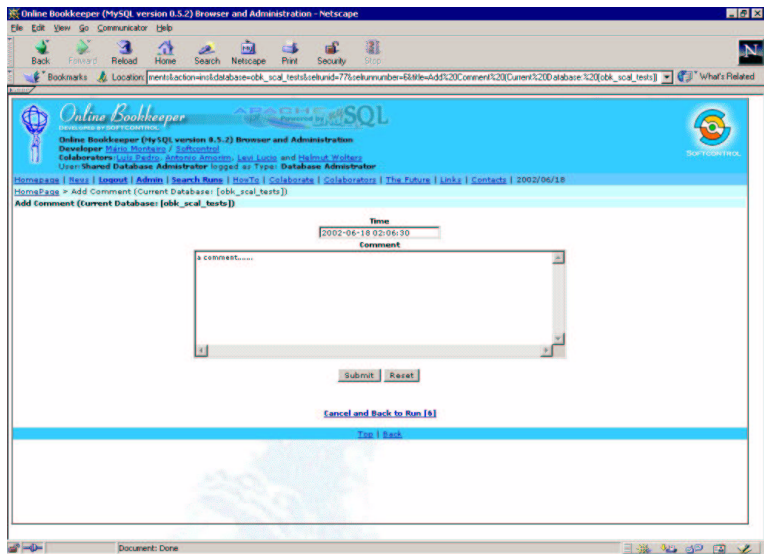


FIGURE 5.4. Adding the comment



As can be seen in figures 5.3. and 5.4., while using the OBK/MySQL the user can add annotations to a run by first performing a search for it (fig. 5.3.), and then clicking on the “add comment” link. He/she will then be presented with the annotation introduction form as displayed in fig. 5.4..

5.1.2. Data browsing

The OBK package offers the user several ways of accessing the book-kept data: command line utilities, an internet database browser and a C++ query library to be linked with client user applications. The next paragraphs provide an overview of these mechanisms - each of them is suited for a different context of usage of the bookkeeping data.

5.1.2.1. Command line utilities

Despite the fact that browsing the OBK data in command line mode is not very sophisticated, for fast checks this is probably the easiest mechanism to use. The

OBK/Objy and the OBK/OKS provide an application that does an ASCII dump of the OBK data to the screen.

The user can choose different types of data to be retrieved by specifying command line arguments to the dump application (see synopsis on Appendix A).

EXAMPLE 5.6.

```
bash$ obk_dump_oks -P be_test_single_host
Run Number: 1000
Start date: 26/10/01
Start time: 21:28:42
End date: 26/10/01
End time: 21:28:42
Status: GOOD
Repetition number: 0
Number of events: 0
Maximum number of events: 0
Recording enabled: 0
Trigger type: 0
Detector mask: 0
Beam type: 0
Beam energy: 0
```

In example 5.6. one can see how to printout the summary information for each of the runs stored for the “be_test_single_host” partition. The example uses the *obk_dump_oks* (OBK/OKS) application.

5.1.2.2. Web-based browser

For each one of the OBK implementations a web-based database browser was built. Browsing the book-kept data using a web client (e.g. Netscape or IExplorer) has the advantage of being simple and accessible from everywhere in the world (the ATLAS collaboration is spread around the globe).

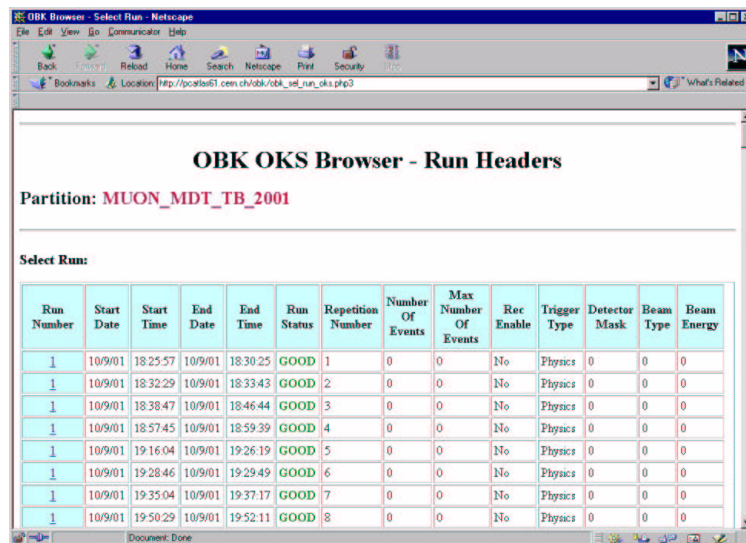
The first OBK web-based browser was built for the OBK/Objy. Being a first try, the browser is naturally quite simple, allowing in the first screen the selection of

5.1. The OBK from the user's viewpoint

the partition, in the second the selection a run inside the partition and in the third the visualization of all the bulked data about the chosen run.

The second internet browser was built for the OBK/OKS and includes more advanced features than its predecessor, namely the display of run summary information and of IS meta-information. In particular, the display of run summary information (see fig.5.5.) was a major step from the first browser which only showed run data in bulk. The implementation of this useful feature came to be after receiving feedback from the people using the OBK/Objy internet browser in the 2000 testbeam.

FIGURE 5.5. OBK/OKS web-based browser - Displaying run summary information



OBK OKS Browser - Run Headers

Partition: **MUON_MDT_TB_2001**

Select Run:

Run Number	Start Date	Start Time	End Date	End Time	Run Status	Repetition Number	Number Of Events	Max Number Of Events	Rec Enable	Trigger Type	Detector Mask	Beam Type	Beam Energy
1	10/9/01	18:25:57	10/9/01	18:30:25	GOOD	1	0	0	No	Physics	0	0	0
1	10/9/01	18:32:29	10/9/01	18:33:43	GOOD	2	0	0	No	Physics	0	0	0
1	10/9/01	18:38:47	10/9/01	18:46:44	GOOD	3	0	0	No	Physics	0	0	0
1	10/9/01	18:57:45	10/9/01	18:59:39	GOOD	4	0	0	No	Physics	0	0	0
1	10/9/01	19:16:04	10/9/01	19:26:19	GOOD	5	0	0	No	Physics	0	0	0
1	10/9/01	19:28:46	10/9/01	19:29:49	GOOD	6	0	0	No	Physics	0	0	0
1	10/9/01	19:35:04	10/9/01	19:37:17	GOOD	7	0	0	No	Physics	0	0	0
1	10/9/01	19:50:29	10/9/01	19:52:11	GOOD	8	0	0	No	Physics	0	0	0

The third and most recent OBK internet browser was built for the OBK/MySQL prototype. This version is much more sophisticated than the previous ones, featuring not only data browsing, but also database administration, helpful utilities, user access control (see 5.1.3) and the possibility of adding annotations directly in the browser (as mentioned before in 5.1.1.4.). Given the interest of the ATLAS community in this tool, the implementation of graphic files upload for complementing

annotations and the implementation of an additional mechanism for querying IS data are also previewed.

5.1.2.3. C++ *query library*

Despite the previously mentioned browsing mechanisms, it was felt by the OBK users that a programmatic query interface for the book-kept data was necessary. This interface allows users to automate their browsing requests by writing C++ code that retrieves and parses the data to fit specific needs.

Despite its usefulness, the C++ query library was only built for the OKS prototype. It remains however a good study on relevant book-kept data and how to present that data in programmatical form. In table 5.5. an overview of the library's methods is provided. All the detailed information may be found in the "OBK/OKS API user's Manual" [32] (appendix E).

TABLE 5.5. OBK query library methods

<code>list<string> * getPartitionNames</code>	retrieve all the partition names in the database
<code>list<string> * getRunNames</code>	retrieve all the run names for a given partition
<code>OBKRunHeader * getRunHeader</code>	retrieve summary information for a given run
<code>OBKParameterValue * getParameterValue</code>	retrieve all the values an parameter of a given IS object assumes during a given run
<code>OBKISInfo * getISInstance</code>	retrieve all the information about a given IS object in a given run
<code>list <OBKComment> * getComments</code>	retrieve all the annotations performed on a given run

The return values for the methods are either simple or composed C++ STL [33] data structures (classes). These return data structures (OBKRunHeader, OBKPa-

parameterValue, OBKISInfo and OBKComment) include methods for easy looping through and retrieving the data.

In order to use the query functionality, the user will have to link his/her C++ programs against the *obkqueryoks.so* shared library which is distributed with the OBK software.

5.1.3. Utilities

During the evolution of the OBK package several problems which came up justified the development of special applications (utilities) for dealing with them. Obviously, for the most advanced OBK/MySQL prototype the amount of these utilities is much larger than for the first OBK/Objy. The next few lines start by describing what are the utilities which are common to all OBK implementations, then the ones that are common to the OBK/OKS and the OBK/MySQL and finally the ones that are only available for the OBK/MySQL.

All implementations:

- delete runs: in the OBK/Objy and the OBK/OKS a command line application exists for this purpose. For the OBK/MySQL it is possible to do it using the web-based browser, although special user privileges are required.

OBK/OKS and OBK/MySQL:

- delete partitions: as for delete runs;
- release a locked partition: one of the problems common to all the OBK implementations concerns the termination of the acquisition process via the KILL signal. While acquiring data the OBK keeps a lock on the partition on which it is running. If the process is not given a chance to remove this lock before finishing, no other OBK process may be started on that partition. The OBK/OKS implements a command line application to remove locks manually, while the OBK/MySQL allows doing it through the web-based browser (for users with adequate permissions).

OBK/MySQL:

- Define actions to be done in the beginning of a run: by default the OBK acts as a passive receiver to the MRS and IS messages it subscribes to when the process starts. A feature requested after the 2001 testbeam was the possibility to define, per partition, the names of IS objects whose values should be explicitly read at the beginning of a run. The OBK web-based browser allows this parameter input to users with adequate permissions (fig. 5.6.);

FIGURE 5.6. Definition of names of IS objects to be read at start of a run

Online Bookkeeper (MySQL version 9.5.2) Browser and Administration - netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Stop

Location: [n_actions.php?action=ins&id=obk_scal_test&subpartition=2&node=actions&file=Create%20Action%20Current%20Database%20jobk_scal_test%20] What's Related

Online Bookkeeper
Developer: Mario Huchler / mario.huchler@cern.ch
Collaborators: Luis Padua, Roberto Andreatti, Leo Lodi, and Roberto Walther
User: Shared Database Administrator logged as Type: Database Administrator

Home Page | Home | Logout | Admin | Search Items | Home | Collaborate | Collaborate | The Future | Links | Contacts | 2002/06/10

Home Page > Create Action (Current Database: [jobk_scal_test])

Create Action (Current Database: [jobk_scal_test])

Action Name:

Object Name:

Start Time:

End Time:

StoreConfDB:

[\[Cancel and Back to Actions\]](#)

[Top](#) | [Back](#)

Document: Done

- Enable/Disable acquisition of run configuration data: the OBK/MySQL is capable of storing run configuration data (parameters fed into the several detector and DAQ systems before the acquisition starts). Through the web browser it is possible to enable or disable this feature (obviously only to users with sufficient permissions);
- User management: since the OBK/MySQL web-based browser allows both read-only and write (administration) actions to be performed on the database, a

user privilege mechanism had to be devised to prevent abuse. Through the browser it is possible to create new users according to certain predefined user types which grant different privilege levels.

5.1.4. Available functionality per prototype

Table 5.6. provides a summary of the available functionality per prototype in order to clarify the evolution of the OBK and to somewhat resume section 5.1.. The features that are not clear from previous descriptions are explained in the list following the table:

TABLE 5.6. Functionality per prototype

	Run Status	Repeated Runs	Web Browser	C++ query library	Release Lock	Store IS Meta-data	Store run config.	Retrieve IS on-demand
OBK/Objy	YES	NO	YES	NO	NO	NO	NO	NO
OBK/OKS	YES	YES	YES	YES	YES	YES	NO	NO
OBK/MySQL	YES	YES	YES	NO	YES	YES	YES	YES

- **Run Status:** during data acquisition not all the runs terminate correctly. If a run finishes in error state (for instance, beam is lost) then the data may not be valid. The OBK keeps a flag signaling the correct or faulty end of run;
- **Repeated Runs:** during production it is expected that an unique run number will be provided to all of the DAQ. During development however, the different subsystems may want to use the DAQ for independent testing purposes - in this case the run number may be repeated. The OBK detects this situation and prevents the overwriting of existing data.

5.2. *Implementation issues*

In the section that follows, a technical description of the three OBK prototypes is provided. For each of the implementations the relevant class diagrams are presented, as well as the schematics representing the physical database structure. Moreover, the workings of each of the web-based browsers are discussed and specific technical characteristics of each one of the prototypes are explored.

First, however, in order to place the technical description under perspective, the tools and languages used to build the prototypes are mentioned and briefly discussed. After the description of the prototypes the text also refers the platforms for which the OBK builds.

5.2.1. *Languages and tools*

The following is a list of used languages and tools while developing the several OBK prototypes:

- C++ programming language: this extremely well-known programming language [34] is used all across the TDAQ for core software development. All the OBK acquisition engines and user API software are written using C++;
- STL (Standard Template Library): the standard template library is a set of customizable data container and algorithm templates which can be used as building blocks for C++ applications;
- Objectivity/DB: Objectivity/DB is a distributed object oriented database management system [35]. It allows interfacing with standard languages (C++, Java, Smalltalk and SQL) in order to manage the persistent objects;
- OKS: an in-memory persistent object manager implemented to satisfy the needs of the ATLAS TDAQ in terms of configuration databases. The strength of the OKS lies in being lightweight and oriented to clients which pose strong efficiency and real-time requirements;

- MySQL: open source relational database product known worldwide for its speed and ease of use. MySQL also provides interfacing with the C++ language by means of an API;
- PHP: a widely-used general-purpose scripting language that is especially suited for Web development [36]. All the web-based browsers developed for the OBK make extensive use of this language;
- Perl: another widely-used general-purpose scripting language. Several OBK utilities were developed using this language;
- Apache: a widely-used HTTP web server.

5.2.2. *Objectivity/DB prototype*

As mentioned before, the first OBK prototype made use of Objectivity/DB for persistency. Objectivity/DB provides a C++ binding, which means that the user can define his/her persistent classes using this language. This feature makes the interfacing C++ applications with the databases becomes (almost) transparent. The persistent objects which are managed by Objectivity/DB may be dealt with as any other transient object while in memory, the advantage since they survive in secondary storage after the application ends.

The data model of objectivity consists of four layers of abstraction:

- **Federation**: an Objectivity federation logically contains one or more databases, their corresponding data models and a catalog of all databases inside the federation;
- **Database**: a database contains objects, which may be clustered into containers;
- **Container**: object clustering unit, for efficiency and performance purposes;
- **Object**: persistent object, containing data and methods to deal with that data. At the level of the object (class definition), Objectivity/DB supports a multitude of

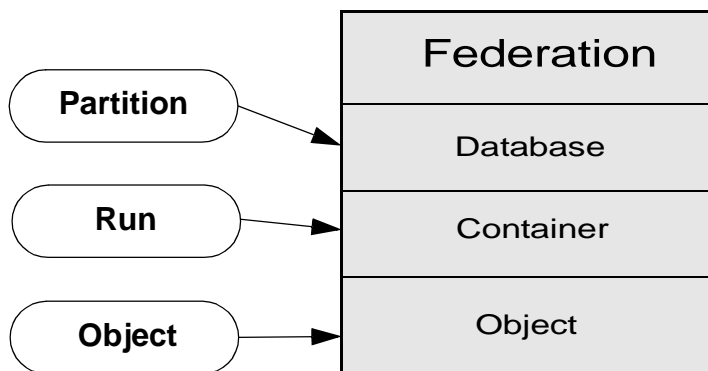
primitive data types, aggregate data types (e.g. arrays), associations between classes, as well as inheritance.

Objectivity/DB also provides locking services (a specialized server) in order to manage concurrency issues. While dealing with multiple clients of a database, one of the issues is to avoid simultaneous writes of the same block of data or reads of data which is being written at that moment. Objectivity/DB provides locking services at the level of the federation, the database and of the container.

A good practice while building a data-centered applications [37] is to start by constructing the data model and then to build the application around it. The description contained in the next paragraphs follows this philosophy, reason why the data model is described first:

The first problem while mapping the OBK logical data model described in fig. 5.1. into the Objectivity/DB data model was how to use advantageously Objectivity's federation/database/container/object layered structure. Fig. 5.7. depicts the adopted solution:

FIGURE 5.7. Mapping of OBK's data model into Objectivity/DB's data model



This mapping option seems quite natural and mainly stems from the fact that clustering all the objects belonging to the same run in a single container improves access to run information, thus benefiting queries that focus on data from a single

5.2. Implementation issues

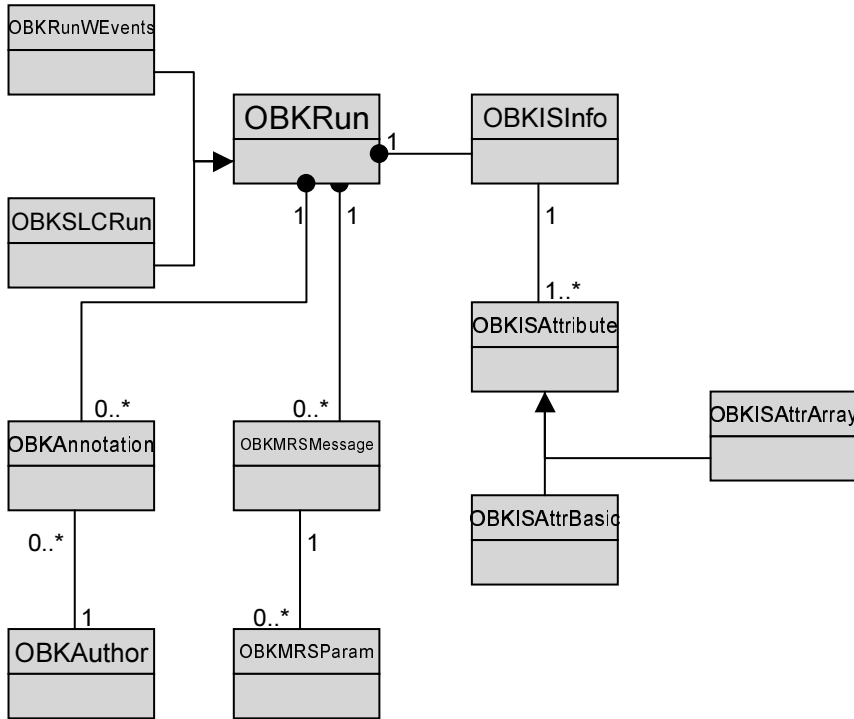
run. It also allows managing the concurrency (locking) at two different levels of granularity: partition or run.

In what concerns detailed data modeling, figure 5.8. presents the Objectivity/DB persistent class information model of the book-kept data. As can be seen from the diagram, the OO concepts of inheritance and composition are used in order to simplify and organize the data model. To be noticed also that, as explained before, all

the persistent objects may become transient during run time. This means that the application may manipulate them as normal⁵ C++ objects.

In Figure 5.8. the persistent class structure for OBK/Objy (i.e. the database schema) is presented:

FIGURE 5.8. Persistent class structure for the OBK/Objy (UML)



As can be seen from the above diagram, the mapping of the bookkeeping data (chapter 4) into a C++ class structure is quite natural and intuitive. The main class is the abstract *OBKRun*, which holds (by composition) all the MRS, IS and annota-

5. In fact persistent objects are managed using the Objectivity provided ooRef or ooHandle wrapper classes, which provide almost transparent usage of C++ objects while enabling them with persistency properties.

tion information. To notice the two classes *OBKRunWEvents* and *OBKSLCRun*, which inherit from *OBKRun* and implement the logical concepts of run with events (Run) and slow control run (SLCRun). The *OBKLockedRuns* class is an auxiliary class used in order to always keep track of the last run which was correctly stored.

As depicted in fig. 5.7., objects of type *OBKRun* are created as containers, which means that all the objects coupled with a run (IS, MRS or annotations) are stored in contiguous memory/disk pages⁶ - access to objects in contiguous memory is very efficient. There is no particular class to implement the logical concept of partition, as the OBK process creates a database with the partition name for each new partition where the data acquisition is started.

The enabling of C++ classes with persistency properties is done via a dedicated Objectivity/DB mechanism which is be used in the following manner [35]:

1. Creation (by hand, using Objectivity/DB tools) of a federated database;
2. Definition of the data model (persistent classes) by writing extended C++ header files containing the class definitions. The extension consists of making persistent classes inherit from the *ooObj* class and replacing pointers with Objectivity/C++ references (see example 5.7.);
3. Processing of the data files defined in step 2 using the *ooddlx* processor provided by Objectivity - from this phase two C++ header files containing the user defined classes and classes required by Objectivity are created. A C++ code file containing part of the implementation of the classes defined in the header files is also created;
4. Implementation of the user defined methods to manipulate the data in the persistent classes and of the application using those persistent classes;
5. Compilation of the hand-written and Objectivity/DB generated implementation files and linking with the necessary libraries.

6. Objectivity/DB uses a page server architecture, which means that the unitary block of exchange between secondary (disk) and primary (main) memory is a page. A page contains a cluster of objects, which acts as a cache and speeds up accesses while avoiding disk accesses.

EXAMPLE 5.7. Definition of the Run persistent class using the Objectivity/C++ DDL⁷

```

class Run : public ooContObj {
public:
    Run();
    Run(uint32 runNumb);
    void setRunNumb(uint32 runNumb);
    void setStartDate(const d_Timestamp& startDate);
    void setEndDate(const d_Timestamp& endDate);

    uint32 getRunNumb() const;
    virtual uint32 getSubRunNumb() const = 0;
    d_Timestamp getStartDate() const;
    d_Timestamp getEndDate() const;
    virtual void print() const;
    void print(uint8 h) const;

    ooRef(OBKComment) runComms[] <-> commToRun[];
    ooRef(Coordinator) runCoordinator <-> rCoordinated[];
    ooRef(LockedStatus) runToLStat[] <-> lStatOfRun[];
    ooRef(OBKConfdb) hasConfig <-> appliesToRuns[];

protected:
    uint32 m_runNumb;
    d_Timestamp m_startDate;
    d_Timestamp m_endDate;
};

```

Inherits from the container Objectivity class => persistency

Objectivity/C++ reference to access persistent objects

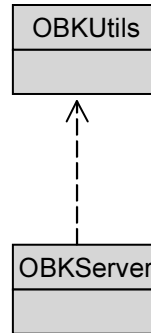
Definition of associations*

* Associations model relationships between objects (i.e. one-to-many, one-to-one, ...). With relationships it is also possible to create composite objects with behavior propagation (delete, lock, copy and versioning).

Having the database schema ready, it is possible to proceed to figure 5.9. which displays the transient classes *OBKServer* and *OBKUtils*, as well as their relationships with the persistent classes in fig. 5.8. (represented by *OBKRun*).

7. Data Definition Language

FIGURE 5.9. Transient class structure for the OBK/Objy (UML)



The *OBKServer* class provides a high level interface to the OBK database. It includes public methods to store MRS messages, IS messages and annotations (*processMRSMsg()*, *processISInfo()* and *createComment()*), to safely stop the acquisition (*stop()* and *abort()*) and to browse the database (*listPartitions()*, *listRuns()* and *printRun()*). The *OBKUtils* class provides additional services for timestamp (from IS and MRS messages) conversions.

Finally, the *OBKServer* class is instantiated by the *obk_daq*, *obk_dump*, *obk_online_comment* and *obk_offline_comment* applications (see appendix A for the list and description of the OBK/Objy binaries). A special mention to the *obk_daq* application, which takes care of subscribing to the MRS and IS servers by using the MRS and IS receiver classes (see 4.1.1.1. and 4.1.1.2.) and passing these messages to *OBKServer* class objects for storage in the database.

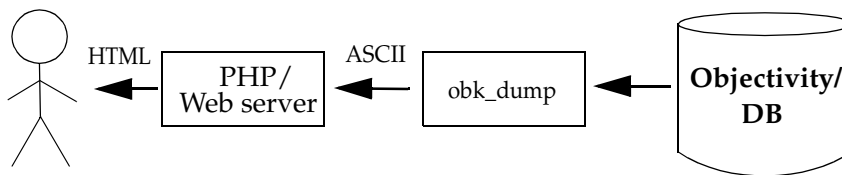
5.2.2.1. Web browser implementation

The web-based browser for the OBK/Objy was developed using PHP scripts to produce the dynamic HTML that shows the data contained in the Objectivity data-

base. To make this data available web-wide, the Apache web server with the PHP interpreter module installed is used.

In what concerns extracting the data from the database, since PHP provides no API to talk to the Objectivity/DB engine, a special application had to be developed for this purpose. In fact, the *obk_dump* binary provides a command line option (-F) to produce formatted ASCII output to be interpreted by the PHP scripts. When executed by Apache/PHP interpreter, the PHP scripts call the *obk_dump* application with the necessary flags in order to access the necessary information, parse it and make it available as HTML. Figure 5.10. depicts this process:

FIGURE 5.10. OBK/Objy's web-based browser topology



5.2.2.2. OBK/Objy Specificities

The OBK/Objy uses specific Objectivity/DB facilities to ease coding of the application while improving the efficiency. Some of these facilities are:

- Transactions: Objectivity/DB works based on a transaction model - a transaction consists of a group of operations that doesn't affect the database until it is finished (committed). This feature is a very important security feature for databases because it allows the safe removal of database operations that didn't finish correctly (rollback), thus avoiding database corruption.

The OBK starts a new transaction for each run, which means that it is possible to safely keep all the previous stored runs if the data acquisition for the current one goes wrong for some reason (e.g. Objectivity/DB server dies due to power failure);

Objectivity/DB provides two sort of commit policies: *Full Commit*, which closes the federated database, releases all locks and resolves all the object references or handles into OIDs⁸; *Commit-and-Hold*, which is a lighter version of Full Commit since it preserves locks and pointers to objects. The OBK/Objy executes a *Commit-and-Hold* at the end of each run and a *Full Commit* when the acquisition process is stopped;

- Concurrency issues: A very simple overview of Objectivity's locking mechanism goes as follows: a basic object can have multiple readers, as long as there is no process writing it. When a process has an update lock on an object, no other process (reader or writer) may use either the object or the container that holds that object (which also gets locked). However, Objectivity/DB makes available a mechanism which allows safe reading of objects which have one update lock - MROW (Multiple Readers One Writer).

The fashion in which the OBK uses these mechanisms is the following: processes which are writing to the database hold update locks for objects being written; reader processes (e.g. *obk_dump*) use the MROW mechanism in order to be able to see all the runs in the database, even the ones that are currently being written.

Apart from preventing accesses to objects which are currently being used, the OBK/Objy also includes a simple mechanism to avoid that two OBK acquisition processes start on the same partition (which could provoke two simultaneous writes of the same data): when an OBK acquisition process starts in some partition it checks a known file (*obk.pid*) to see if any OBK process is already running in that partition. If not, it starts operation and writes to the *obk.pid* file its *pid*⁹ and the name of the partition. If yes, the OBK acquisition process stops.

8. Object Identifier - each persistent object has a unique identifier which allows it to keep its identity when either in disk or in main memory.

9. Process Identifier - in the UNIX or Linux operating systems, each running process holds an identification number.

5.2.3. *OKS prototype*

The second OBK prototype makes use of OKS for persistency. As Objectivity/DB, OKS is also an OO DBMS that provides a C++ binding. However, given that the OKS is an ATLAS homegrown solution, it does not require a commercial licence to be used outside CERN. An OBK prototype based on OKS was found advantageous in the sense that collaborating institutes located all over the world could test and use the book-keeping software, free of charge.

Despite its object-orientation, the OKS tool is quite different from the Objectivity/DB one in many aspects. In fact OKS does not provide an engine to process requests from clients wanting to access the database. It is rather a library to be used in conjunction with user applications, that supports simple persistent in-memory object management. Although in general the data model adopted by OKS is less sophisticated than the Objectivity/DB one (e.g. persistent object do not support embedded methods), OKS provides very fast processing since it is lightweight and the data that can be dealt with at a certain moment is necessarily in main memory.

The OKS data model itself is quite simple: the OKS persistent object manager uses the concept of datafiles - these are containers of data (objects) which are held unitarily in memory and loaded/stored from and to the filesystem in atomic operations. The OKS can manage several datafiles in memory simultaneously, but while in memory a datafile has to be complete. These files are ASCII XML information and may be stored in the local filesystem, AFS or NFS.

At the level of persistent class definition, the OKS library provides roughly the same data types as Objectivity/DB (although defined in different ways), as well as class associations and an inheritance mechanism.

From the description above, it is clear that the OKS data model does not support such a “clean” mapping of partition and run concepts as was seen in 5.2.2. for Objectivity/DB. This problem was obviated by mapping the OBK data into OKS in the way depicted in fig. 5.11.:

FIGURE 5.11. Mapping of OBK's data model into OKS's data model

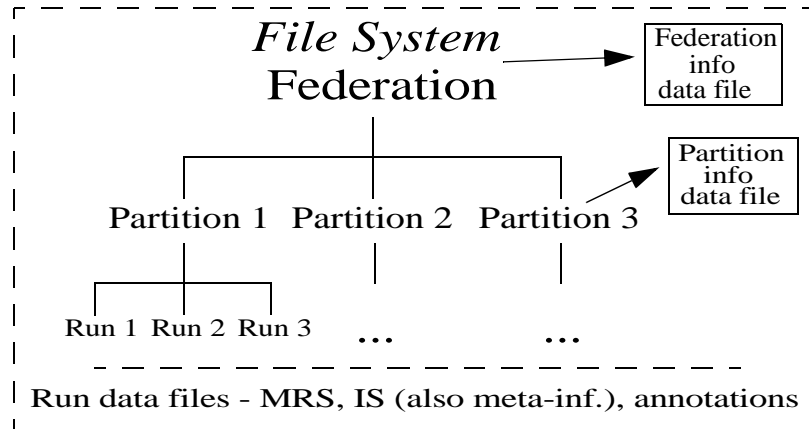
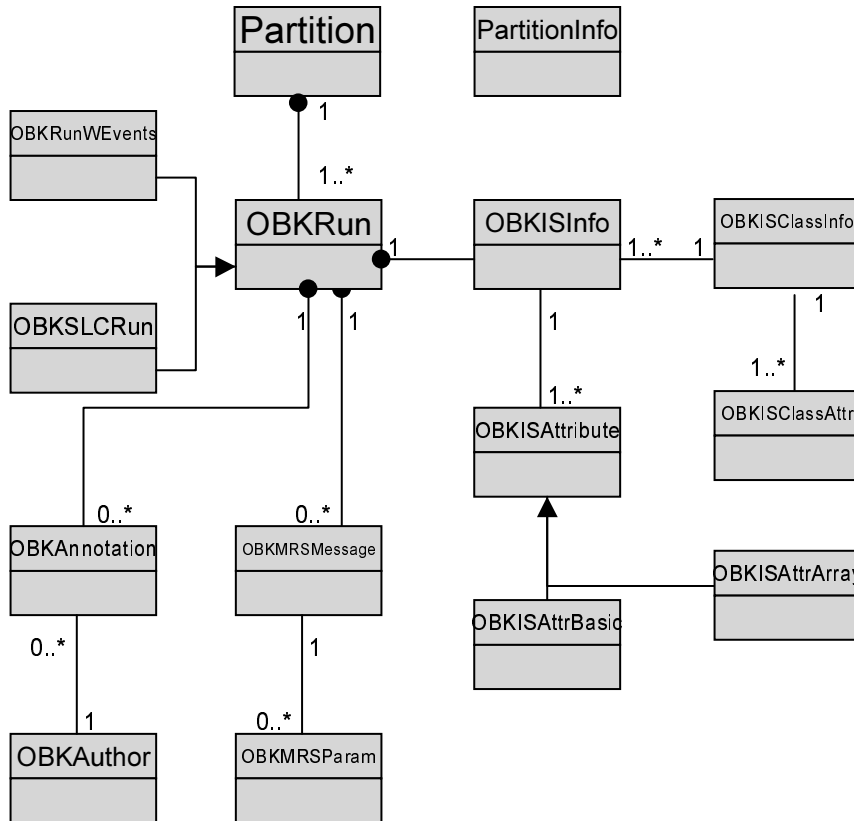


Figure 5.11. basically shows that the OBK/OKS database is a set of data files organized using the filesystem's directory structure: in the first level of the directory structure it is possible to find a data file containing information about the partitions for which book-kept data exists; in the second level a data file containing all the summary run information for the partition coexists with the data files which contain specific MRS, IS and annotation information.

Figure 5.12. displays the persistent class structure for the OBK/OKS prototype. The similarities between this figure and figure 5.8. account for the compatibility of Objectivity/DB's and OKS's data model at the level of class definition.

FIGURE 5.12. Persistent class structure for the OBK/OKS (UML)



The major differences between the OBK/Objy's and the OBK/OKS's persistent class structure are:

- the inclusion of the persistent classes *Partition* and *PartitionInfo*. Objects of type *PartitionInfo* are stored at the level of the federation information data files (see fig. 5.12) and hold the names of all the partitions for which data exists as well as a logical field signaling whether that partition is currently in use or not. In what concerns objects of type *Partition*, these are stored at the level of the partition information data files and contain pointers to objects of class type *Run* which are also stored in the same datafiles.

5.2. Implementation issues

- the inclusion of the *OBKISClassInfo* and *OBKISClassAttrInfo* classes to handle the storage of IS meta-information.

The way OKS provides persistency properties to its data is somewhat different from Objectivity/DB. OKS does not provide persistency to the C++ classes themselves (as does Objectivity/DB, by inheriting from *ooObj* or *ooContObj* classes), it rather uses a defined set of C++ classes contained in the OKS library to build the database schema and to manage the data itself. In this sense the C++ binding of OKS is lighter than the Objectivity/DB one.

The database schema may either be created at run time using the OKS schema classes (*OksClass*, *OksAttribute*, *OksRelationship*) or built previously by a dedi-

cated application and stored as an XML data file which can be loaded by the application requiring it. The OBK/OKS uses the second approach.

The instantiation and usage of OKS objects is done via the OKS data classes *OksClass* and *OksData* as can be seen from example 5.9., where an object of class type *PartitionInfo* is instantiated and its attributes *partitionName* and *inUse* are set.

The schema consists of a number of objects of type *OksClass* which are used as a reference by the *OksObject* constructor¹⁰ to build objects of OKS user-defined type classes. Example 5.8. depicts the definition of the Run persistent class:

EXAMPLE 5.8. Definition of the Run persistent class using OKS

```
OksClass *PartitionInfo = new OksClass( "PartitionInfo",
    "Information one partition in the federation.",
    false);
{
    OksAttribute *partitionName = new OksAttribute(
        "partitionName",
        OksAttribute::string_type,
        false,
        "unknown",
        "String that holds the name of one of the partitions.",
        true);

    OksAttribute *inUse = new OksAttribute(
        "inUse",
        OksAttribute::bool_type,
        false,
        "unknown",
        "Boolean field that indicates whether the partition is in
        use or not.",
        true);

    PartitionInfo->add(partitionName);
    PartitionInfo->add(inUse);
}
```

To notice also in example 5.9. the usage of an object of type *OksKernel* to get a reference to the *PartitionInfo* class type object. *OksKernel* is used for administrat-

10. Each C++ class has one or several constructor methods with which it is possible to initialize the data members of a newly created object or perform necessary operations at object creation time.

ing the whole of the database, including functions such as loading/saving data files to disk, setting verbosity levels or getting database statistics.

EXAMPLE 5.9. Creation and usage of OKS objects

```
OksKernel kernel = new OksKernel();
OksClass *PartitionInfo = kernel->FindClass("PartitionInfo");
OksObject *piData;

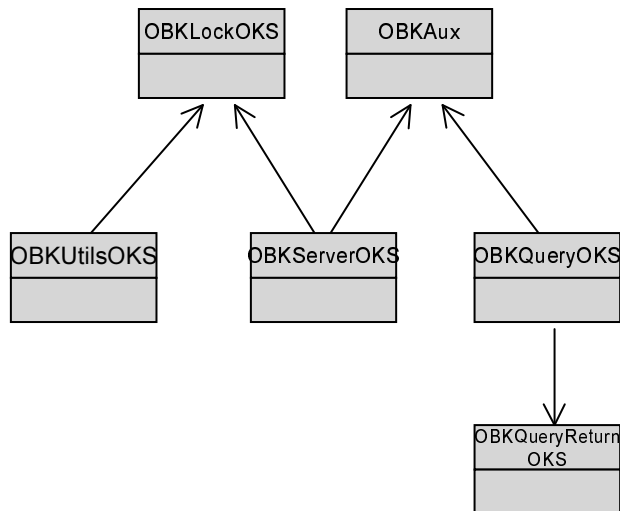
piData = new OksObject(PartitionInfo, partitionName.c_str());

OksData d;
d.Set(partitionName);
piData->SetAttributeValue("partitionName", &d);

d.Set(true);
piData->SetAttributeValue("inUse", &d);
```

The transient OBK/OKS class model can be seen in figure 5.13.. As can be understood while comparing this figure with the transient class model for the OBK/Objy (fig. 5.9.), the OBK/OKS class model splits and groups common functionality in specialized classes.

FIGURE 5.13. Transient class structure for the OBK/OKS (UML)



The *OBKAuxOKS* and *OBKLockOKS* classes are used as auxiliaries, providing support to *OBKServerOKS*, *OBKUtilsOKS* and *OBKQueryOKS*. It needs to be mentioned also that the *OBKQueryOKS* class builds as a C++ shared library which provides a C++ API to the user (see 5.1.2.3.).

5.2.3.1. Web browser implementation

From the implementation point of view, the OBK/OKS web-based browser is almost similar to the OBK/Objy's one. The browser includes better features than its predecessor, along with improved usability (see 5.1.2.2.).

5.2.3.2. OBK/OKS specificities

While building the OBK/OKS, several strategies were adopted in order to deal with the fact that the OKS library does not support many of the features which are included in Objectivity/DB (e.g. concurrency management, transactions). Still other mechanisms were devised in order to deal with other constraints, such as for instance speed of access to the database. Some of these specificities are described in the following points:

- Concurrency issues: As mentioned before, the OKS library does not support any sort of concurrency management. For this reason it was necessary to implement a way in which write exclusivity may be granted to a client. This feature is specially important for OKS since data files are completely loaded to memory - if two clients load the same data file simultaneously for write purposes, the changes done by the first one who saves will not be taken into consideration.

A very simple lock mechanism was implemented in order to deal with this problem: the process which requires the lock reads from a well known file the lock status. If the lock is on (the file contains "1"), the process requiring the lock waits until the lock is free (the file contains "0"). It then acquires the lock by writing "1" to the file. Despite the fact that the operating system provides better solutions, this very simple mechanism solves the problem for the granularity required by the OBK.

The problem of not allowing that two OBK acquisition processes start in the same partition was solved by the *OBKLockOKS* class (see fig. 5.13). This class includes methods that allow the locking and the releasing of one partition. An OBK process that may write to the database is always required to request a lock on the partition it is going to act upon;

- Transactions: the OKS does not support transactions, which means that recovery actions for a failed database update do not exist. The way to guarantee some safety was to store one run (including all necessary messages) per data file(s). This means that if failure occurs during data acquisition, only the current run data is lost (all the previous runs were already stored);
- Optimization of data accesses: since the OBK stores information on a run basis, it is very important to know for each run what type of physics data is being searched for and how (run parameterization). Given the importance of these parameters, it is wise to have them ready for fast access. Keeping this trend in mind, each run's parameters are stored in the partition data file in *Run* class type objects (see figures 5.11. and 5.12.). This strategy drastically diminishes read access times by avoiding the problem of opening the IS messages data files for each run and searching for the messages that contain this sort of information.

5.2.4. MySQL prototype

Finally, the third OBK prototype was developed using the MySQL DBMS [38] for persistency. Being an Open Source product, MySQL does not pose licence problems such as Objectivity/DB while providing widely tested and quality database services. The decision to implement a MySQL based prototype stemmed also from

the desire to try a relational approach while developing the OBK, as opposed to the two previous object oriented approaches.

The MySQL package makes available an engine for dealing with SQL queries¹¹ which when deployed runs continuously, accepting local or remote connections. These connections may be established directly by human users (MySQL client console) or by software applications - MySQL makes available an ODBC and a C/C++ interface.

As mentioned before, MySQL implements the relational model (see 3.2.1.), allowing the creation of databases - each database contains a number of tables and each table contains a number of fields (e.g. strings, numbers, dates). Tables may be associated with other tables, implementing relationships such as one-to-one or one-to-many.

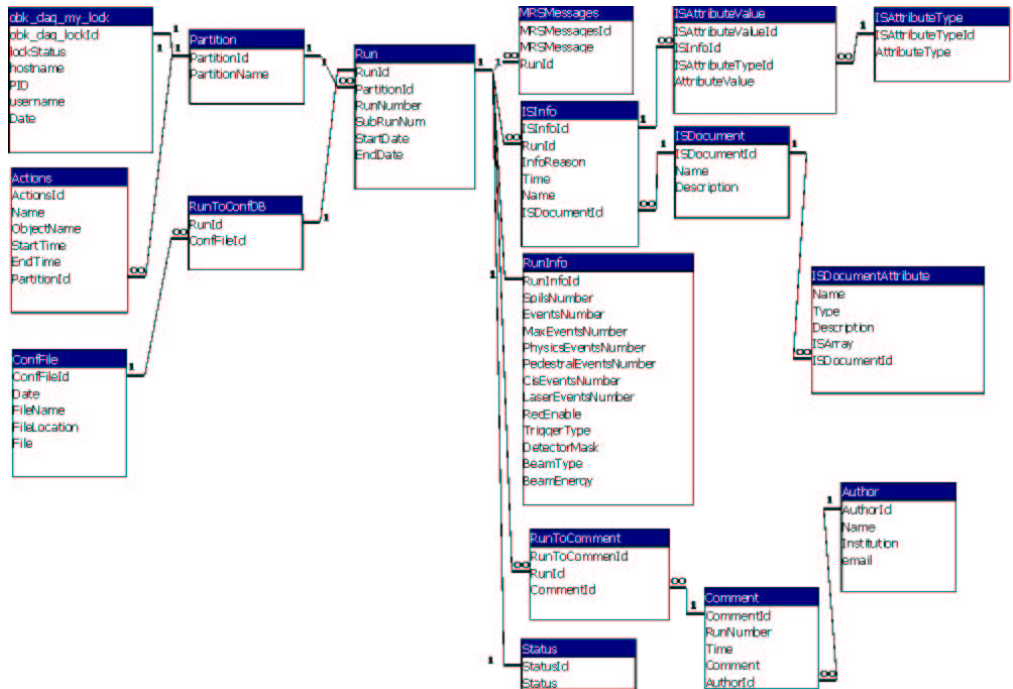
It is easy to understand that the previous mapping of OBK's data using the object oriented data model is not valid while dealing with the relational data model. In fact, as was already discussed in chapter 3, the relational data model is closer to the

11. All the database manipulation operations are performed on the MySQL database through the use of the SQL language.

5.2. Implementation issues

machine than the OO data model which was invented later. For this reason, strong redesigning of the database schema was necessary.

FIGURE 5.14. Relational data model for the OBK/MySQL



As can be observed in figure 5.14., the OBK/MySQL persistent class structure varies substantially from the structures implemented for the previous prototypes. In wide terms, the main concepts (e.g. Run, IS and MRS messages) which were previously described as classes are now mapped into tables. The relational result is however less elegant since the data model is less expressive - take for instance the need to use the *RunToConfDB* or *RunToComment* tables to implement many-to-many relationships.

Apart from data model differences, it is also possible to notice in fig. 5.14. newly introduced concepts:

- the *actions* table which defines objects which value should be read from IS at start of run (see 5.1.3.);
- the *conffile* table which stores the configuration files which are associated to a run. Since the configuration files are stored by OKS in XML (text) format, it is possible to record them in a table field of type *text* (ASCII stream).

An example of usage of MySQL through the C/C++ API is given out in 5.10.. As can be observed, the C API makes available a series of functions and data struc-

5.2. Implementation issues

tures which are used to sent out queries to the MySQL engine and to process the results of those queries.

EXAMPLE 5.10. Example of usage of the MySQL C API

```
MYSQL *sock,mysql;
MYSQL_RES *res;
MYSQL_ROW tmp;

int RunId = 0;
string selectqbuf;
unsigned int nFields = 0;

selectqbuf = ("SELECT RunId FROM run WHERE RunNumber = " +
             intToString(runNumber) + " AND SubRunNum = " +
             intToString(SubRunNumber) + " and PartitionId = " +
             PartitionId);

if(mysql_query(sock,selectqbuf.c_str())) {
    userMessaging->m_obkErr(new string("Query: " + selectqbuf + "
                                     failed! " + (string)mysql_error(sock)),2);
}

if(!(res = mysql_store_result(sock))) {
    userMessaging->m_obkErr(new string("Couldn't get result from
                                     query: " + (string)mysql_error(sock)),2);
}

nFields = mysql_affected_rows(sock);

if(nFields == 0) {
    RunId = 0;
}
else {
    tmp = mysql_fetch_row(res);
    RunId = atoi(tmp[0]);
    mysql_free_result(res);
}
```

MySQL data structures definition

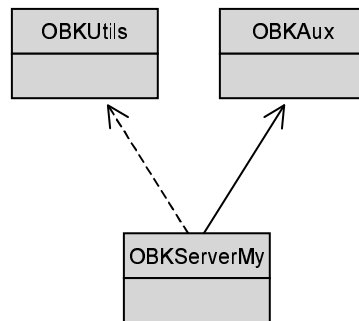
Query execution

Retrieving the query's result

The transient class structure for the OBK/MySQL is presented in figure 5.15.. It is somewhat similar to the transient structure of the OBK/Objy, as it was imple-

mented based on that first OBK prototype. This fact is mainly due to the shortage of time and manpower available at the time of implementation.

FIGURE 5.15. Transient class structure for the OBK/MySQL



5.2.4.1. Web browser implementation

The implementation of the web-based browser for the OBK/MySQL is quite diverse from the ones of the previous prototypes. Instead of using a specialized application (see fig. 5.10) to retrieve data from the database, the browser communicates directly with the MySQL engine using an already existing interface in native PHP.

This approach has great advantages given that it allows the direct passing of SQL queries from PHP to the MySQL engine, making the search very flexible. This possibility was not implemented in the previous prototypes given the fact there are no interfaces between PHP and MySQL for OKS or Objectivity/DB.

5.2.4.2. OBK/MySQL specificities

- Concurrency issues: the OBK/MySQL includes locking mechanisms to avoid either having either two writers or one reader and one writer trying to access the

same data. However, from the previous prototypes it was already shown that it is necessary to avoid the possibility of launching two OBK data acquisition processes simultaneously. The OBK/MySQL implements this by writing to a specific table (`obk_daq_my_lock`) who started an OBK process on which partition. Before starting book-keeping a partition, each OBK acquisition process reads the `obk_daq_my_lock` table to check if there is an OBK process already started on that partition. If not, the process starts. If yes, it stops;

- Transactions: despite the fact that MySQL supports transactions and atomic operations, the OBK/MySQL does not make use of that possibility. While the previous prototypes consider a run to be a transaction unit, the present one does not support that feature. If there is a failure, the OBK/MySQL stores the data that was introduced in the database until that failure occurred. Although there is no major risk of damage for the OBK database, this problem should be better addressed in future releases of the OBK/MySQL;
- Optimization of data accesses: as for the OBK/OKS prototype, the OBK/MySQL prototype keeps the run parameters in a special purpose container - the `runinfo` table. The principle is the same, allowing fast accesses to parameters which are the most important while describing the run;
- Storage of configuration data: as mentioned before, the OBK/MySQL's schema holds the `conf` table which is used to store the run configuration data. Configuration for each run is composed of a number of text files, which are copied entirely into the MySQL database in case they don't already exist there (several runs may share the same configuration files).

5.2.5. *Supported platforms*

The platforms¹² involved in the data acquisition are an heterogeneous collection, reason why the Online Software builds (at the time of the writing of this dissertation) for the following ones:

- 2 Linux platforms: Red Hat 6.1/egcs-1.1 and Red Hat 7.2/gcc-2.96
- 2 Sun platforms: Sun 2.5.1/egcs-1.1 and Sun 2.7/CC-5.2
- A Lynx platform (real time Operating System): Lynx 3.0.1/g++-2.9-98r2

The OBK only builds for the Linuxes and the Suns, given that the Lynx machines are dedicated to run real-time constrained physics data acquisition.

The presented list of platforms varied substantially during the project, change that was due to the evolution of operating systems and compilers over the two years the project lasted. As mentioned before in 2.3.2., to handle the multiple changing platforms in a relatively automatic way, a set of tools was used:

- CVS (Concurrent Versions System) [39] to centrally keep track of changes to the source code. CVS also allows several developers working simultaneously on the same package by warning the developer when his/her version of the source code is outdated in comparison to the one held in the repository;
- SRT (Software Release Tools) [40] to manage the differences while building for different platforms. Instead of requiring a makefile for each different platform (taking into consideration that the system libraries may be located in different places, the compiler options may be different, etc), the SRT provides the facilities to build an abstract specification of the software to build and how to install it. Details particular to each platform are solved automatically by the tool. SRT is a homegrown solution from CERN.

12. Operating System + C++ compiler version

- CMT (Configuration Management Tool) [41] performs the same function as the SRT tool. Developed in collaboration with CERN, this software provides more functionality than SRT. The Online Software counts on adopting CMT and drop SRT in the near future, as SRT is no longer supported. While this dissertation is being written, the Online Software builds using both tools.

In order to check the constant integration (at the level of the programming interfaces) of the several packages of the Online Software, a nightly build is setup. The build job, which is launched automatically, gets the needed sources from CVS, builds them for each specific platform according to the rules defined by SRT (or CMT) and installs the needed binaries and libraries. When a release is to be built the procedure is the same, only not done automatically since the software needs to be first checked for correct building and running.

5.3. *Summary*

The developed OBK software can be described from two different viewpoints: the user's and the developer's. From the user's perspective, the system can be divided into the data acquisition and the browsing module. The data acquisition module acquires information into the OBK database which is logically organized into partitions. Partitions contain runs which in turn contain IS messages, MRS messages, Comments (run annotations), IS meta-information and Configuration Data. The user may define the behavior of the OBK prototypes by means of environment variables which are different for each of the prototypes (Objectivity/DB, OKS, MySQL). The OBK can be started either with the rest of the Online Software (by a specialized script) or in standalone mode from the command line. Either while in data acquisition or offline, the user can annotate runs by executing command line applications for that purpose or by using the OBK web-based browser (depending on the OBK prototype). Another aspect of the usage of the OBK is browsing recorded data. This can be done in three different ways: through a command line application, through a web-based browser or by means of a C++ API.

From the developer's viewpoint several aspects of the OBK may be considered: the prototypes were built using a varied set of languages and tools, such as C++, STL, Objectivity/DB, OKS, MySQL, PHP, Perl and Apache. Despite the fact that all the OBK prototypes are based on the same conceptual model, each implementation is different given the distinct database technologies and APIs they use. Due to this trend and also to the fact throughout the project knowledge about design problems evolved, each OBK prototype is distinct from the other. This fact can be faced as providing an extra degree of richness for the evaluation phase of the present dissertation. It needs to be mentioned also that the OBK prototypes are built for several different platforms, all either Suns or Linuxes.

Evaluation of the practical work

The present chapter focuses on evaluating the practical work done in the context of this dissertation. It concludes the software development process cycle by describing the test and integration phases, as well as discussing some issues regarding deployment and metrics.

Section 6.1. lays down the functionality, performance and scalability tests that were performed on each of the OBK prototypes. The integration of the OBK with other Online Software components is also discussed.

In 6.2. deployment is addressed. The real-life situations where the OBK was used are described along with the gathered results and feedback into the development.

6.3. presents and comments on metrics for the various phases of the software development process of the OBK.

Finally, section 6.4. provides a discussion on lessons learnt throughout the whole development process of the OBK software.

6.1. Test and integration phase results

In this section the tests performed on the several OBK prototypes are described and explained. Firstly the functionality and error recovery tests are mentioned, followed by the performance and scalability ones. It is important to refer that all the tests obeyed to a common test plan [42], which allowed the easy comparison of results. After describing the tests and interpreting the results, the issues regarding integration with other Online Software components are discussed.

6.1.1. Functionality and error recovery tests

Functionality tests were performed on the OBK to evaluate its conformity against the requirements. As is described in the test plan document, the following are the several functionality tests that were applied to each prototype:

1. check if the OBK correctly connects to all the specified sources of data;
2. check if OBK archives all the MRS and IS messages received as well as the selected information from the configuration database. Check if all the information is stored in the proper run and if the run parameters are correctly collected;
3. check if OBK properly stores information on abnormal end of run;
4. check if OBK properly stores operator's comments;
5. check OBK behavior when can't establish connection to the database server;
6. check OBK behavior if loses the connection to the database server;
7. check OBK behavior when an existing run number is given;
8. check OBK behavior when trying to use an invalid MRS/IS server;
9. check OBK behavior when a MRS/IS server crash;
10. check OBK behavior if more than one obk acquisition process is started (in the same and in different partitions).

Special purpose testware was developed in order to perform the above mentioned tests. The results can be found in the Test Report document [43] (also found in appendix G). The functionality tests yielded valuable input by either pinpointing

flaws on the software or asserting its correction, thus increasing the level of confidence on the OBK package.

6.1.2. Performance and scalability tests

The following lines will describe the performance and scalability tests that were performed on each of the OBK prototypes. A discussion of the results will also be provided. The text is based on the results presented on the “Test Report of the Online Book-keeper for ATLAS DAQ Online Software” document [43], as well as on the first paper written in the context of this dissertation [44] (appendix C) and presented in Beijing, China, at the Computing in High Energy Physics (CHEP 2001) conference.

6.1.2.1. Test definition

The scalability and performance tests can be divided into two major groups:

Data acquisition

1. Calculate the mean time to store typical MRS messages;
2. Calculate the mean time to store typical IS informations;
3. Calculate the mean time to store the selected information from the configuration databases;
4. Test the behavior of OBK’s data acquisition application with concurrent access from multiple IS and one MRS servers;
5. Calculate the disk space required to store a predefined amount of book-keeping data.

Data retrieval

6. Calculate the mean time to retrieve all the information about a particular run;
7. Calculate the mean time to select all the runs that fulfill a given criteria (explained later);

Special testware applications were also built in order perform the scalability and performance tests. These applications were used in conjunction with instrumented

OBK code in order to produce timing results. The obtained results are presented in the next section.

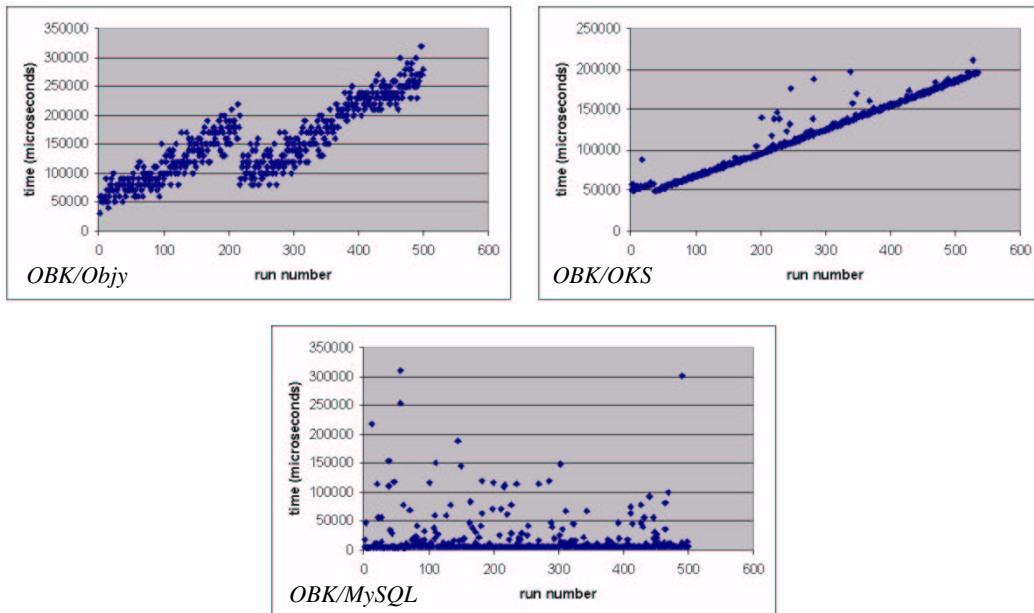
6.1.2.2. Test results

Data acquisition

The results presented in this section are based on the following premises:

- All the test messages are sent without any delay in between;
- The machines where the tests were performed (linux7.1/gcc2.96 - PIII/800MHz) were unloaded;
- The test databases are stored to: a remote server machine running the Objectivity/DB server in the case of OBK/Objy; the local filesystem in the case of OBK/OKS; a remote server machine running the MySQL server in the case of OBK/MySQL.

FIGURE 6.1. Test 1 results - time to store an MRS Start of Run message



6.1. Test and integration phase results

FIGURE 6.2. Test 2 results - time to store a typical IS message

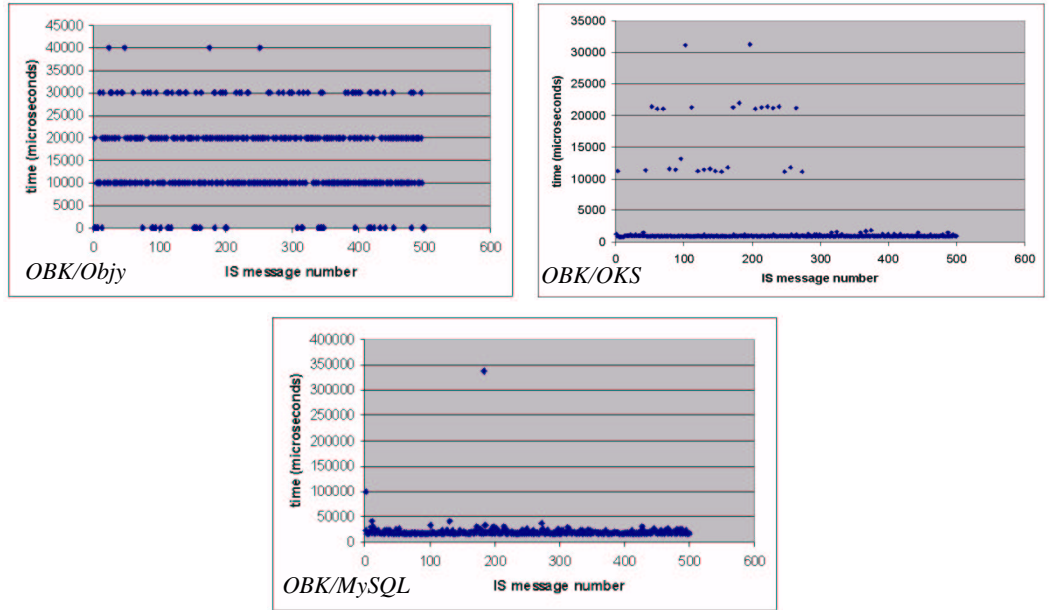


FIGURE 6.3. Test 3 - time to store an MRS Start of Run message + config. data

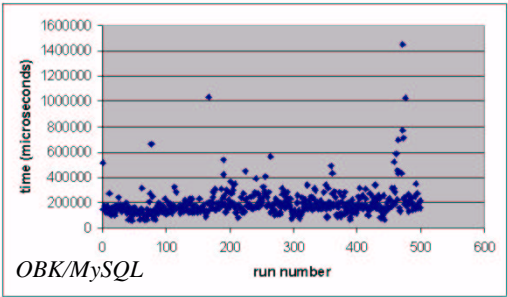


TABLE 6.1. Test 4 - concurrent access from multiple IS servers + 1 MRS server

prototype vs # of IS servers	10	20	50	100
OBK/Objy	OK	OK but some- times the IS/ MRS servers become blocked	Online software becomes blocked	-
OBK/OKS	OK	OK	OK but some- times the IS/ MRS servers become blocked	Online software becomes blocked
OBK/MySQL	OK	OK but some- times the IS/ MRS servers become blocked	Online software becomes blocked	-

TABLE 6.2. Test 5 - disk space required to store a sample OBK database (1000 runs)

OBK/Objy	OBK/OKS	OBK/MySQL
63.4	3.7	2.11

6.1. Test and integration phase results

Data Retrieval

The results presented in this section are based on the following premises:

- The machines where the tests were performed (linux7.1/gcc2.96 - PIII/800MHz) were unloaded - in tables 6.3. and 6.4.;
- *local* means the query application and the database server are located in the same machine; remote means the query application is located in a different machine from the database server, on the same LAN;
- The databases used for the tests are filled with 1000 runs;
- The criteria for the query on test 2 is met by 5% of the runs;
- For the tests performed on the OBK/OKS, the remote case applies to the writing of the database files through NFS (network filesystem).

TABLE 6.3. Test 6 - time to retrieve all the information about a single run

	Local	Remote
OBK/Objy	0.13	0.94
OBK/OKS	0.38	1.42
OBK/MySQL	0.39	0.70

TABLE 6.4. Test 7 - time to select all the runs that fulfill a given criteria

	Local	Remote
OBK/Objy	18.13	116.08
OBK/OKS	0.35	1.18
OBK/MySQL	0.02	0.08

6.1.2.3. Test results discussion

The following lines provide an interpretation of the results presented in the previous section, along with a final comment which tries to wrap-up the results and

present a conclusion involving all the prototypes. For each one of the prototypes, only the relevant results will be mentioned.

OBK/Objy

- **Test 1:** The OBK/Objy presents the steepest rise in time to create a new run. This problem can be attributed to the longer times it takes to create a new container as the database grows bigger, since the OBK checks before whether a run with the same number already exists. One can also see in figure 6.1. a clear decrease of the times at run ~200. This takes place when the OBK/Objy is terminated and a full commit occurs;
- **Test 5:** The disk space taken by the OBK/Objy is one order of magnitude higher than for the OKS and MySQL implementations. This time is mainly dominated by the size of the memory page defined when the federation is created - a container always allocates a predefined number of pages [45];
- **Test 6:** Only one container is opened, outperforming the other prototypes when the test is executed locally;
- **Test 7:** In this second query the OBK/Objy presents the worst performance, both in the local and in the remote case. This trend can be attributed to the fact that the information the query searches for is cached (kept in special containers) by both the OKS and the MySQL prototypes. On the other side, the OBK/Objy has to download all the containers in the database to find the required information.

OBK/OKS

- **Test 1:** As for the OBK/Objy, this prototype also presents a clear rise in the time it takes to create a new run. The reason for this problem is also the same, i.e. the raise in the time to check whether a run with the same number already exists;
- **Test 2:** While storing IS informations the OBK/OKS is actually the fastest prototype. Being an in-memory DBMS, the OKS only performs I/O operations to

secondary memory either to create, load or store data files. Since access to data files only happens at the beginning and end of run, the storage of IS informations (predominating over MRS messages) is done in-memory, which justifies the very low times - taken up mainly by processing operations;

- **Test 4:** Also in this test the OBK/OKS outperformed the other prototypes. The test involved starting a variable number of IS servers, feeding them with continuous information and letting the OBK subscribe to all that information. The OBK/OKS is capable of coping with a larger flux of IS data, which is due to the fact that it processes IS informations faster than the other OBK prototypes. In fact, the IS servers are not fully asynchronous with its receivers (e.g. OBK), given that the messages are held in a queue while waiting for processing. If the processing on the receiver side does not cope with the requests, the queue is filled and the IS servers block;

OBK/MySQL

- **Test 1:** The OBK/MySQL is the only prototype which presents a stable time value to start a new run. Despite the fact that the code checks whether the run number already exists or not, this does not seem to affect the performance;
- **Test 3:** One can observe a very large difference in the times to store a start of run message in fig. 6.1. and 6.3. (configuration data storage off or on, respectively). Clearly, storing the configuration data files places a heavy processing burden on the OBK;
- **Test 5:** As can be seen in table 6.2., the OBK/MySQL presents the lowest occupied space for the same amount of book-keeping data. This trend seems to be due to the fact that while for the MySQL prototype all the data is stored in a fixed number of tables (each table corresponding to a file), for the OKS prototype 4 data files (with all its overheads) are created for each run;
- **Tests 6 and 7:** While performing data retrieval queries the OBK/MySQL presents the best performance results. The queries for the OO databases are built using C++ code for browsing through the persistent objects, while for the

MySQL database the query is done via the SQL language. It is clear from the tests that the SQL processing engine is more optimized than the hand-written queries used for the OKS and Objectivity/DB book-keeping databases.

From the test results one can withdraw the following generic conclusions concerning each prototype: the better overall performance was achieved by the OBK/MySQL; the OKS implementation of the OBK also presents strong results which are mainly due to its in-memory features; the OBK/Objy presents the less optimal results. This last trend can be attributed mainly to the facts that the OBK/Objy was the first prototype to be designed and that Objectivity/DB requires deep know-how in order to be tuned correctly. In [45], a study in optimization of the OBK/Objy is provided.

6.1.3. Connectivity to other Online Software components

As was mentioned in chapter 4, the OBK has connections to the IS, MRS and Configuration Databases components of the Online Software. Given that the ConfDB data is accessed directly via the filesystem (not through a specialized server), the connection to the ConfDB component cannot affect the performance of the Online Software.

However, after analyzing the results in table 6.1. it is possible to understand that the performance of the OBK also influences the performance of the IS and the MRS¹ servers to which are subscribed. In case of overflow of the OBK parts of Online Software system may eventually slow down considerably or even crash, since the whole of the Online Software depends on the correct functioning of the messaging system.

Test 4 should however be put under perspective, since the probability of the need of the OBK to subscribe to the amount of servers tested is very low. It is nonetheless indicative that the callback times for the OBK should be kept as low as possible.

1. Given that the subscription mechanism is similar.

6.2. Deployment

The OBK was deployed for both large scale tests within the Online Software and testbeams (sub-detector test) taking place every year during the summer. These tests involved the Online Software running as a whole, which allowed the detection of problems in the OBK that would have not arisen in a very controlled testing environments such as the ones described in sections 6.1.1. and 6.1.2..

6.2.1. Large scale tests (of the Online Software)

For the finalized ATLAS experiment, the Online Software system is supposed to serve an indeterminate number of clients requiring control, configuration and monitoring services. At the time of the writing of this dissertation, the final number of nodes requiring Online Software services is estimated at several thousands. This being, it became necessary to verify the functionality of the system at these scales. The tests were done accordingly to a specified test plan [46].

In 2001 the Online Software the tests were performed correctly on 111 controlled nodes, running on 111 PCs. In 2002 the tests were extended to 210 machines, running 210 controllers. Although it was possible to go up to ~600 nodes running on the 210 machines, instability was encountered and this scale.

The OBK was used to create and browse the log book for these tests - no scalability problems were encountered. Some minor functionality bugs were discovered and fixed both in 2001 (OBK/OKS) and 2002 (OBK/MySQL). The tests were useful mainly as experience in setting up and configuring the OBK to run in an integrated fashion with the rest of the Online Software. Results of the tests including comments on OBK usage can be found in [47].

6.2.2. Testbeams

The ATLAS is composed of several sub-detectors, each one of them being separately tested for correct functioning. At the beginning of 2000 the DAQ/EF-1 pro-

totype (including the Online Software) was adopted as the data acquisition software for the Tilecal detector. In the years that followed the Muon Drift Chambers and the Pixel subdetector also made the same choice. For the 2000 testbeam the OBK/Objy was used, for 2001 the OBK/OKS and for 2002 the OBK/MySQL.

The testbeams provided excellent testbeds for the Online Software, given that the nodes being controlled have real interactions with the detector and are not simulated, as happens in the large scale tests described in the previous sections. In what concerns the OBK the testbeam experience was extremely fruitful: large databases were created, allowing to check the scalability of the tool when the amount of book-kept data becomes significative. The next is a list of problems (apart from the functionality ones) and requests identified by users during the several testbeams in which the Online Software participated:

2000 testbeam (OBK/Objy):

- Too simplified web browser - need to present the data in a more readable fashion and not in “bulk”;
- Overall Online Software slowdown due to communication problems with Objectivity/DB server;
- Request for an API to retrieve the book-kept data via a programming language.

2001 testbeam (OBK/OKS):

- The filesystem approach to store the database generates too many files in a not centralized fashion. This leads to easily losing track of the book-kept data;
- Request for redesign of certain parts of the OBK/OKS C++ API;
- Request for implementation of a mechanism that allows the active retrieval of IS data by the OBK at the beginning and end of run.

*2002 testbeam*² (*OBK/MySQL*):

- Slow SQL queries on large databases (hundreds of megabytes);
- Request for increased functionality for the browser in terms of data display.

2. In progress at the time of the writing of this dissertation.

6.3. Some metrics

In the first part of this section some metrics concerning the several phases of the software development process of the three OBK prototypes are presented. The numbers should be interpreted loosely (except for the lines of code metrics), given that most of the time the several tasks were done in parallel and with a varying number of developers. In the second part (table 6.7) a match is made between the original learning contract (appendix B) and the achieved goals during study time.

- **Requirements gathering:** 2 man/month of pure requirements gathering. The “Online Book-keeper Requirements” document [30] was produced. Despite the fact that time was dedicated specifically to requirements gathering, this was a task that spanned throughout all the project given the continuous input from OBK’s users;
- **Design and implementation:**

TABLE 6.5. Design and implementation metrics

	Effort (man/month)	Lines of Code (LOC)
OBK/Objy	18	4166
OBK/OKS	4	8799
OBK/MySQL	3	6193

- **Testing**

TABLE 6.6. Testing metrics

	Effort (man/month)
OBK/Objy	1
OBK/OKS	1
OBK/MySQL	0.5

- **Documentation:** approximately 3 man/month were used in writing the OBK requirements document [30], the OBK user’s guide [31] and the OBK test

6.3. Some metrics

report [43]. This time also includes the writing of the online documentation (e.g. release notes) included in the Online Software's web page [1].

The remaining time was used in maintenance of the package. This included tasks such as bug fixing, integration with the release tools, preparation for scalability tests and test beams and answering questions from the users.

TABLE 6.7. Achieved goals from the original learning contract

Code	Original objective	Secondary (de facto) objective	Documented in
A	Multiple readings about the databases discipline and the concrete book-keeping problem.		Chapters 2 and 3
B	Research on solutions for similar problems, tools and methodologies.		Chapter 3
C	OKS port of the book-keeper prototype.		Chapters 4 and 5, Appendixes E and F
D	Testing of the OKS prototype and comparison with the Objectivity/DB one.		Chapter 6, Appendixes C, D and G
E	Familiarization with the ConditionsDB tool.	Familiarization with the MySQL DBMS tool.	
F	Refinement of the requirements, design and methodology used in the previous prototype. ConditionsDB port of the OBK.	Refinement of the requirements, design and methodology used in the previous prototype. MySQL port of the OBK.	Chapters 4 and 5, Appendix F
G	Evaluation of the new MySQL prototype.		Chapter 6, Appendixes C, D and G
H	Comparison of all the developed prototypes and mapping of solutions/results within the previous research.		Chapter 6, Appendixes C and D
I	Evaluation of the database subject in general and recommendation of necessary resources for a production OBK.		Chapters 6 and 7, Appendixes C and D

6.4. Lessons learnt and practical recommendations

The next paragraphs address the conclusions that can be withdrawn from the experience acquired while developing the OBK software. These conclusions are also (partly) mentioned in the second paper written in the context of this dissertation [48] (appendix D) which was presented at the Very Large Databases (VLDB 2002) conference which took place in Hong-Kong, China.

Software Development Process

Following a clear software development process was a very useful and enriching experience. As was mentioned in chapter 2, following the same set of guidelines while building each prototype made clear where are the differences between the three implementations. That was not the only advantage, as one can observe in tables 6.5 and 6.6 that the effort diminished both in the design & implementation and testing phases for the latter prototypes. The acquired experience along with the previously written documentation and the automatization of certain tasks (e.g. testware for the test and integration phases) lead to an increasing optimization of the whole process.

From a software engineering point of view this project provided a way of understanding in the field the benefits of a formalized approach to the development of software, specially in the case of large applications such as the Online Software (~1 million LOC). The clear approach to phases that typically receive less attention (requirements, high level design, testing & integration and documentation writing) simplified and helped to the delivery of a quality OBK product. Also, it was well understood that the usage of auxiliary software engineering tools and

templates (e.g. CVS, SRT, CMT, documentation templates) induces a clean and effective software developing environment.

Technology

The necessity for integration of the various tools used while developing the OBK package provided valuable experience in what concerns the construction of multi-technological software.

- ***DBMS technology:*** the OO paradigm was found to be more flexible for mapping the OBK data than the relational one. In fact, while the database schemas for the OBK/Objy and the OBK/OKS are quite natural and easily capture the real-world entities the OBK stores, more trouble was found while using the relational data model. OBK/MySQL's schema (see 5.2.4.) is less intuitive than the others due to the more rigid (tabular) data model approach. In particular, the lack of collection types (e.g. arrays, lists) was felt.

Another plus for the OO DBMS approaches is the elegance of the code, as both for the OBK/Objy and the OBK/OKS the close integration of the C++ language with the data model yields code which is uniform, easy to design and to read. To be mentioned also that if one takes into consideration the difference in functionality for the different implementations, the amount of code for the several prototypes is similar (probably slightly less in the case of the OBK/MySQL).

On the other side, MySQL presents a very strong alternative in terms of performance (see 6.1.2.2.). Relational databases are presently the dominating solutions in the market due to the state of maturity of the technology. SQL engines are very efficient, result of ~30 years of investigation and deployment;

- ***Evolution of the design:*** while within the context of an online system, a strong requirement for the OBK is light weight in terms of processing and primary memory usage. To fulfill this purpose the design evolved in terms of minimizing accesses to the database while keeping in memory the least possible amount of data. In what concerns data retrieval, it was found that using caches for frequently accessed data drastically reduced read times;

- ***Use of XML:*** In the OBK/MySQL, XML strings were used to cope with the difficulty of storing collection types (parameters which vary in number), not available in the data model implemented by MySQL. Despite not being too elegant, this technique seems to simplify storage of data during acquisition and could even be used in conjunction with the OO prototypes to avoid the creation of too many simple data objects;
- ***Tools:*** as was already mentioned CVS, SRT and CMT contributed to a better software developing environment. PHP was found to be a very flexible tool for web development, specially if used in conjunction with MySQL. Perl [49] was used mainly as “glue” between applications for tasks such as testing, integration or database error recovery software.

Interaction with users

Although it may seem a commonplace, it is important to state that a good and constant interaction with the final users of the system being built yielded simpler and faster development. In the OBK project this trend was particularly evident, given that the interaction with the users grew as the project advanced in time. Enhanced knowledge about the systems with which the OBK is connected to as well as increasingly better interaction with the people working on those systems helped putting the project under perspective.

In fact, while for the first prototype(s) the development was done based only the first set of requirements stated in [29], for the latter ones the day-to-day and test-beam feedback proved to be extremely valuable in understanding thoroughly the problem at hand. This facilitated focusing on the aspects of the OBK which are most relevant and the tuning of the usage of the software development process accordingly. To be mentioned also that from this continuous interaction with the users a formal open requirements document for the OBK [30] came to be.

The available means were used in order to ease communication between OBK developers and users: meetings (live and by phone), email, visits to the testbeam site, informal corridor talks, etc. As a result of this effort, at the time of the writing of this dissertation the OBK package is being actively used by the following

6.4. Lessons learnt and practical recommendations

groups at and outside CERN: Tilecal subdetector people (CERN), Muon subdetector people (CERN), TDAQ Dataflow people (CERN) and Brookhaven Lab. people (USA).

6.5. *Summary*

In order to assert the soundness of the OBK software, a set of functionality/error recovery and performance/scalability tests was prepared and applied to the OBK. While the functionality and error recovery tests provided the means of correcting functionality bugs, the performance and scalability tests allowed a deep analysis of the software both in terms of data acquisition and data retrieval. After analysis of the results, it is possible to attribute the best overall performance to the OBK/MySQL, which seems to be in general faster and more stable than the other proto-types. The OBK/OKS also presents good overall results and the OBK/Objy seems to be the less optimal solution. In what concerns the connectivity with the other Online Software components, it was observed that the processing times of the OBK affect the IS and MRS components and should thus be kept as low as possible.

From 2000 until the time when this dissertation is being written, the OBK software was deployed consecutively for two types of situations: large scale tests of the Online Software or testbeams (tests involving real beam and parts of the ATLAS detector). In what concerns the large scale tests, they involved deploying the Online Software to control in a simulated environment an increasing amount of the TDAQ (up to 210 machines). The OBK behaved correctly and created the log books for the tests. The testbeams implied usage of the OBK in a real life situation, to book-keep several parameters coming from all the TDAQ chain. The interaction with the users yielded extremely valuable input and lead to uncover new requirements for the OBK.

The two years spent developing the OBK software were rich in lessons in software development. In terms of software engineering it became clear that a formalized approach to software construction (defined software development process, usage of auxiliary software engineering tools, good documentation) produces high quality software that is easily maintained and modified. In what concerns the technology, it was found that while the OO paradigm (OBK/Objy and OBK/OKS) is more expressive and yields more elegant design and code than the relational one. On the other side, the relational engine used (MySQL) is very efficient in query process-

6.5. Summary

ing. Finally, despite being common knowledge, it is important to restate that strong and constant interaction with the software's final users is extremely important in order to produce software that is clearly designed and easy to use, as well as to avoid development pitfalls hence accelerating the development.

Evaluation of the research

Chapter 7 concludes the present dissertation by evaluating the developed research work. The results which were presented in chapter 6 are matched and compared with the current directions of the database community.

In section 7.1 considerations about adequate DBMS technology for the OBK are formulated. These considerations take into account not only the results presented in chapter 6, but also the knowledge and insight acquired while attending conferences (CHEP and VLDB) and reading specialized books and articles.

Section 7.2 describes some projects which can be considered scope-related to the OBK.

Finally section 7.3 provides pointers towards work which, if developed, would provide interesting data to further the studies contained in this dissertation.

7.1. DBMS technology for the OBK

In the next paragraphs pointers towards a DBMS technology for the OBK are provided. They are divided into pointers coming from the developed work and pointers coming from the database community in order to facilitate the matching of the two. A final recommendation is provided in the end.

7.1.1. Pointers from the developed work

- As was shown in chapter 6, the OBK prototype which was implemented using MySQL proved to perform generally better than the OO prototypes. It actually seemed to work better than the OO prototypes “out of the box”, without any major tuning;
- Although the OO data model is more complete and flexible than the relational one, the data stored by the OBK is found not to be very complex. Using techniques such as storing complex objects in XML strings the data can be easily accommodated in a relational schema;
- The usage of either in-house developed solutions or open source products seems to be more adequate to the DAQ environment than commercial products. Although CERN possesses licences for DBMS tools such as Objectivity/DB or Oracle 9i, the usage of these DBMSs by institutes outside CERN (without licenses) is difficult.

7.1.2. Pointers from the database community

- While there is still indecision within the physics community on whether to keep on using pure OO DBMS products or to use relational products with OO features, the database community has already moved on and seems to show little or no interest in the pure OO paradigm for databases. Looking back into the 1990s, there was definitely a clear divergency from the relational paradigm. However, the big players such as Oracle, IBM or Sybase were capable of incorporating

the features offered by OO DBMSs in their products, while keeping the power of the already existing relational engines;

- Pure research on databases nowadays has as main concerns either traditional topics (e.g. query optimization, indexing, tuning), or topics that derive from the needs of e-business which seems to be a decisive factor in database evolution. e-business imposes strong requirements on databases at the level of distribution, responsiveness, security or recovery. More importantly than that, the massively distributed system which is today's web includes a myriad of devices which run on different bandwidths, processing power and memory capabilities. Running smoothly poses enormous problems of filtering, routing and generically manage the data flow so that the entire system does not degrade. Putting part of the intelligence in the data itself (self describing information) seems to be the way to go in order to make the complexity manageable.

This being, the database research community is currently highly committed to investigation in storing, querying and indexing XML data, which is thought to be the enabling technology of the next Web revolution of data management [50]. Other research issues such as security and privacy, self-tuning, technology for web search engines, management of enterprise data, data mining or data warehousing are also either related to or stem from the needs of e-commerce;

- Current research as explained in the previous point seems to take into consideration that the foundation of the database field is the relational model [51]. Despite the fact that big database manufacturers offer OO features in their products, the underlying paradigm continues to be the relational one and the new technology is being built on top of that.

7.1.3. Recommendations

From the discussion above, one can reach the conclusion that the DBMS tool which will provide persistency services to the OBK should meet the following set of criteria:

- It should implement either the relational or object/relational paradigm, as this is clearly the direction that the database community is taking and the most techno-

logically advanced database products are either purely relational or object/relational. As was proven by the practical work, the relational data model is sufficient to express the complexity of OBK's data;

- A product developed outside CERN is preferable to an in-house solution. Until the ATLAS experiment is running some years will still pass and the technology which is being used now will undoubtedly present better performances. In order to cope with this evolution it is easier to use an external DBMS package than to maintain and develop a homegrown solution. This also makes it easier to follow the generic direction of the database community and to adopt technology upgrades that may appear;
- A light, fast and simple DBMS system is preferable given the soft real-time characteristics of the OBK software. Heavy DBMSs may comport too many features which are unnecessary for the OBK and may contribute to the slow-down of the data acquisition;
- An open source product is preferable to a commercial one given that the OBK software is to be distributed to several institutes which may not want to buy licenses for commercial products.

7.2. Related work

Inside ATLAS two systems which are very close to the OBK are the Liquid Argon¹ book-keeper [52] and the Tilecal² book-keeper [53]. Both of them store run catalog data and use as underlying technologies PHP and MySQL.

Another interesting book-keeping tool being used in ATLAS, this time for cataloging and replication of distributed data, is the Magda project [54]. Magda is currently being used by the ATLAS's Offline software systems in order to keep track of event data generated in a simulation context. Technologies used include MySQL, Perl, Java and C++.

Finally, although the spitfire project of the DataGrid Data Management Work Package [55] is not a book-keeping tool on its own, it aims at providing middleware for the integration of heterogeneous SQL based metadata³ repositories. Spitfire's purpose is to act as an intermediary between clients of metadata in very large distributed systems. Underlying technologies include (among others) Java, XML and MySQL as the default database backend.

-
1. A sub-detector of the ATLAS, specialized in observing certain characteristics of the physics events.
 2. Idem.
 3. OBK's data can be characterized as metadata, as it is "data which makes data more accessible to the user" [48].

7.3. *Future work*

Given that time and other resources available for the present dissertation were limited, some of the work that could have led to deeper investigations and results was identified but not developed. The following points describe what remains to be done:

- Study on how the OBK prototypes scale in terms of storage of large quantities of data (up to the terabyte order of magnitude);
- Study on how desirable it is to use Object/Relational features in an OBK prototype. This sort of study can be carried out by implementing an OBK which uses Oracle 9i for persistency (Oracle 9i licenses are available at CERN);
- Study on the usage of the OQL query language in the case of Objectivity/DB and of the OKS query API in the case of OKS to understand if these mechanisms produce faster results than the direct manipulation of persistent objects.

7.4. Summary

While in chapter 6 conclusions about the practical work were laid out, the present chapter matches those results with the current directions of the database field. The practical investigation points to MySQL, an externally developed relational DBMS systems as the best of all the tested persistency systems. The database community seems to have always been sceptical about the OO paradigm - all the major players in today's database arena implement essentially relational engines and the current research seems to take into consideration that the foundation for the database field is the relational model. For this reason the recommendation of this dissertation concerning DBMS resources for a production OBK focuses on a light, fast, open source relational DBMS system. In what concerns future work, the most important point is to understand how the several OBK prototypes scale to store larger loads of data than the ones which were already tested.

Glossary

AMS - Advanced Multithreaded Server

API - Application Programming Interface

ATLAS - A Toroidal LHC ApparatuS

CASE - Computer Assisted Software Engineering

CERN - European Organization for Nuclear Research

DAQ - Data AcQuisition system

DBMS - Database Management System

DCS - Detector Control System

DDL - Data Definition Language

EF - Event Flow

HEP - High Energy Physics

HLT - High Level Trigger

HPSS - High Performance Storage System

IS - Information System

LHC - Large Hadron Collider

LVL1 - Level 1 Trigger

LVL2 - Level 2 Trigger

MRS - Message Reporting System

MSS - Mass Storage System

OBK - Online Book-Keeper

OBK/Objy - Online Book-keeper prototype implemented using Objectivity/DB for persistency

OBK/OKS - Online Book-keeper prototype implemented using OKS for persistency

OBK/MySQL - Online Book-keeper prototype implemented using MySQL for persistency

ODMG - Object Data Management Group

OKS - Object Kernel Support

OO - Object Oriented

OODBMS - Object Oriented Database Management System

ORDBMS - Object Relational Database Management System

ROD - Read Out Driver

TDAQ - Trigger and data acquisition system

UML - Universal Modelling Language

VLDB - Very Large Database

References

References

- [1] ATLAS's Online Software home page: <http://atlas-onlsw.web.cern.ch/Atlas-onlsw>
- [2] ATLAS High-Level Triggers, DAQ and DCS - Technical Proposal. CERN, 2000
- [3] Partitioning issues in DAQ/EF prototype -1 - D. Francis and T. Wildish, CERN 1997 <http://atddoc.cern.ch/Atlas/postscript/Note060.ps>
- [4] Object Databases in Practice - Mary E. S. Loomis and Akmal B. Chaudhri, Prentice Hall, 1998
- [5] OKS Documentation Set - Igor Soloviev, CERN 1998
<http://atddoc.cern.ch/Atlas/postscript/Note033.ps>
- [6] Fundamentals of Database Systems - Ramez A. Elmasri and Shamkant B. Navathe. Addison-Wesley, 1994.
- [7] The UNIVAC 1100 in the Early 70s - George Gray.
URL: <http://www.cc.gatech.edu/gvu/people/randy.carpenter/folklore/v1n4.html>
- [8] A General Purpose Programming System for Random Access Memories", C.W. Bachman et al, Proc FJCC 26(1), AFIPS (Fall 1964)
- [9] Hierarchical Database Model & IMS Concepts from University of Pittsburgh.
URL: <http://www.cs.pitt.edu/%7Echang/156/14hier.html>
- [10] A Relational Model of Data for Large Shared Data Banks - E. F. Codd. Communications of the ACM, Vol. 13, No. 6, June 1970, pp. 377-387.
- [11] Third-Generation Database System Manifesto - The Committee for Advanced DBMS Function. Proceedings of the IFIP TC2 Conference on Object Oriented Databases, 1990.
- [12] ODMG-93: The Object Database Standard - Francois Bancilhon and Guy Fer-

ram. SIGMOD, 1993.

[13] Advanced Database Technology and Design - Mario Piattini and Oscar Diaz. Artech House, 2000.

[14] Building a Multi-Petabyte Database: The RD45 project at CERN - Jamie Shiers. Object Databases in Practice, Prentice Hall 1998, pp. 164-176

[15] Jefferson Lab Mass Storage and File Replication Services - Ian Bird et al. Proceedings of CHEP 2001, pp. 276-279

[16] The BaBar Experiment's Distributed Computing Model - Dominique Boutigny. Proceedings of CHEP 2001, pp. 280-283

[17] Object Persistency for HEP Data Using an Object-Relational Database - Marcin Nowak et al. Proceedings of CHEP 2001, pp. 272-275

[18] The ROOT Object I/O System - Rene Brun and Fons Rademakers.
<http://root.cern.ch/root/InputOutput.html>

[19] Critical Database Technologies for High Energy Physics - David M. Malon and Eduard N. May. Proceedings of the 23rd VLDB Conference in Athens, Greece, 1997

[20] Design of the Run Bookkeeper System for the ATLAS DAQ prototype -1 - Antonio Amorim and Helmut Wolters, CERN, 1997.
<http://atddoc.cern.ch/Atlas/postscript/Note049.ps>

[21] The OMG's CORBA website - <http://www.corba.org>

[22] Inter-component communication in the the ATLAS DAQ back-end software - Sergei Kolos, CERN, 1996. <http://atddoc.cern.ch/Atlas/postscript/Note003.ps>

[23] Inter Process Communication package - Serguei Kolos, CERN, 2001.
<http://atddoc.cern.ch/Atlas/postscript/Note075.ps>

References

- [24] Applications of Corba in the Atlas prototype DAQ - Antonio Amorim et al, 10th IEEE Real Time Conference, Beaune, France, 1997
- [25] Design of the Message Reporting System for the ATLAS DAQ prototype -1 - Doris Burckhart, Mihai Caprini, Serguei Kolos and Zuxuan Quian, CERN, 1997. <http://atddoc.cern.ch/Atlas/postscript/Note032.ps>
- [26] Information Service for the ATLAS DAQ prototype -1 - Mihai Caprini, Pierre-Yves Duval, Robert Jones and Serguei Kolos, CERN, 1997. <http://atddoc.cern.ch/Atlas/postscript/Note031.ps>
- [27] Design of the Configuration Databases for ATLAS DAQ prototype -1 - Robert Jones, Michele Michelotto, Ashruf Patel and Igor Soloviev, CERN, 1997. <http://atddoc.cern.ch/Atlas/postscript/Note030.ps>
- [28] Data Access Library for ATLAS DAQ prototype -1 Configuration Databases - Igor Soloviev, CERN, 1997. <http://atddoc.cern.ch/Atlas/postscript/Note054.ps>
- [29] Antonio Amorim and Helmut Wolters, “Requirements for the Run Book-keeper system for the ATLAS DAQ prototype -1”, CERN, 1997. http://atddoc.cern.ch/Atlas/DaqSoft/document/URD_27.html
- [30] A.Amorim, L.Lucio, L.Pedro, A.Ribeiro, H.Wolters, “Online Book-keeper Requirements”, CERN, 2002.
- [31] User’s Guide of the Online Book-keeper for the Atlas DAQ Online Software - Levi Lucio, Antonio Amorim, Luis Pedro and Andre Ribeiro, CERN 2002 <http://atddoc.cern.ch/Atlas/postscript/Note176.ps>
- [32] OBK/OKS API user’s Manual - Levi Lucio, CERN 2001 <http://atddoc.cern.ch/Atlas/Notes/172/Note172.pdf>
- [33] The C++ Standard Library - A Tutorial and Reference - Nicolai M. Josuttis. Addison Wesley, 1999.

-
- [34] C++ Primer - Stanley B. Lippman and Josee Lajoie. Addison Wesley, 1998.
- [35] Objectivity Technical Overview - Objectivity, Inc..
- [36] The PHP website - <http://www.php.net>
- [37] UML for Database Design - Eric J. Naiburg and Robert A. Maksimchuk. Addison Wesley, 2001.
- [38] The MySQL homepage - <http://www.mysql.com>
- [39] The CVS homepage - <http://www.cvshome.org>
- [40] The SRT homepage - <http://atddoc.cern.ch/Atlas/DaqSoft/sde>
- [41] The CMT homepage - <http://www.lal.in2p3.fr/SI/CMT/CMT.htm>
- [42] Test Plan of the Online BookKeeper for the Atlas DAQ Prototype-1 - Antonio Amorim, Levi Lucio, Luis Pedro and Andre Ribeiro, CERN 2002
<http://atddoc.cern.ch/Atlas/postscript/Note167.ps>
- [43] Test Report of the Online Book-keeper for the Atlas DAQ Online Software - Levi Lucio, Antonio Amorim, Luis Pedro and Andre Ribeiro, CERN 2002
<http://atddoc.cern.ch/Atlas/postscript/Note177.ps>
- [44] Experience using different DBMSs in prototyping a Book-keeper for ATLAS' DAQ software - L.Lucio, N.Parrington, A.Amorim, R.Jones, L.Mapelli, L.Pedro,

References

A.Ribeiro et al. Proceeding of CHEP (Computing in High Energy Physics) conference, Beijing 2001, pp. 248-251.

[45] OBK - An Online Book-keeper for High Energy Physics Experiments - Andre Ribeiro. Master dissertation, Faculdade de Ciencias de Lisboa, 2002, pp. 95.

[46] Scalability and Performance Integration Test Plan of the Online Software of ATLAS, CERN 2000. <http://atddoc.cern.ch/Atlas/postscript/Note163.ps>

[47] Systems Integration and Deployment in Test Beams and Large Scale Tests - Online Software team, CERN 2002.

[48] OBK - An Online High Energy Physics' Meta-Data Repository - L.Lucio, N.Parrington, R.Jones, L.Pedro, A.Amorim, L.Mapelli, A.Ribeiro et al. Proceed-

ings of the 28th VLDB (Very Large Databases) conference, Hong Kong 2002, pp. 920-927.

[49] Programacao em Perl (Programming Perl) - L.Lucio and Vasco Amaral. FCA Portugal, 2001.

[50] Data Routing Rather than Databases: The Meaning of the Next Wave of the Web Revolution to Data Management - Adam Bosworth. Proceedings of the 28th VLDB (Very Large Databases) conference, Hong Kong 2002, pp. 3.

[51] Foundation Matters - C.J. Date. Proceedings of the 28th VLDB (Very Large Databases) conference, Hong Kong 2002, pp. 4-5.

[52] ATLAS Tile Calorimeter Run Information Database - Igor Solovianov.
<http://tileinfo.web.cern.ch/tileinfo/runinfo.php>

[53] Bookkeeping Database Search Interfaces - Solveig Albrand and Jerome Fulachier. <http://largbookkeeping.in2p3.fr>

[54] Magda - Manager for Grid-based data - Torre Wenaus.
<http://atlassw1.phy.bnl.gov/magda/info>

[55] Grid Enabled Relational Database Middleware - Wolfgang Hoschek and Gavin McCance. Presented at the Global Grid Forum, Frascati, Italy.

References

*List of binaries, libraries
and scripts for each
OBK prototype*

A.1. OBK/Objy

Available binaries:

- obk_daq (C++ application)
- obk_dump (C++ application)
- obk_offline_comment (C++ application)
- obk_online_comment (C++ application)
- obk_delete_run (C++ application)

Synopsis:

obk_daq

```
obk_daq -p partition-name [-M MRS-subscribe-expression]
        [-n server-name] [-I IS-subscribe-expression]
        [-S Confdb-schema-file] [-D Confdb-data-file]
```

Options/Arguments:

-p partition-name	partition name
-M MRS-subscribe-expression	MRS subscribe expression.
-n server-name	list of IS servers.
-I IS-subscribe-expression	list of IS informations subscribe expression.
-S Confdb-schema-file	Configuration databases schema file.
-D Confdb-data-file	Configuration databases data file.

Description:

This program stores, in an Objectivity database, subscribed MRS messages, ISinformations, and the configuration databases. Default MRS subscribe-expression is ALL.

The number of IS subscribe-expression has to match the number of servers.

obk_dump

```
obk_dump [-p partition-name] [-R run-number] [-F]
```

Options/Arguments:

```
-p partition-name  partition name.  
-R run-number      run number  
-F                 turn on 'php' formatted output
```

Description:

This program dumps the contents of the database. With no arguments, lists the partitions in the database.

obk_offline_comment

```
obk_offline_comment -p partition-name
```

Options/Arguments:

```
-p partition-name  partition name
```

Description:

This program stores, in an Objectivity database, comments to runs.

obk_online_comment

```
obk_online_comment -p partition-name
```

Options/Arguments:

```
-p partition-name  partition name
```

Description:

Send MRS messages to be collected by obk_daq.

obk_delete_run

```
obk_delete_run -p partition-name
```

Options/Arguments:

```
-p partition-name  partition name
```

Description:

This program deletes the selected run and associated sub-runs.

A.2. OBK/OKS

Available binaries and scripts:

- obk_daq_oks (C++ application)
- obk_dump_oks (C++ application)
- obk_offline_comment_oks (C++ application)
- obk_online_comment_oks (C++ application)
- obk_delete_oks (C++ application)
- obk_rebuild_header_oks (Perl script)
- obk_release_lock_oks (C++ application)

Available libraries:

- libobkqueryoks.so (C++ dynamic library)

Synopsis:

obk_daq oks (same as obk_daq)

obk_dump oks (same as obk_dump)

obk_online comment oks (same as obk_daq)

obk_offline comment oks (same as obk_daq)

obk_delete lock oks (same as obk_daq)

obk_rebuild_header oks.pl

```
perl obk_rebuild_header oks.pl location_of_schema_file
                                location_of_federation
```

Options/Arguments:

location of OBK/OKS's schema file
location of the federation

Description:

Rebuilds the header file "partition.data" for a given federation.

obk_release lock oks

```
obk_release_lock oks -p partition-name
```

Options/Arguments:

```
-p partition-name  partition name
```

Description:

This program releases a lock on a database partition that has been left incorrectly locked after processing.

A.3. *OBK/MySQL*

Available binaries:

- `obk_daq_my` (C++ application)

Synopsis:

`obk_daq_my` (same as `obk_daq`)