ATLAS Computing

Technical Design Report

Issue:	1
Revision:	3
Reference:	ATLAS TDR--017, CERN-LHCC-2005-022
Created:	20 June 2005
Last modified:	4 July 2005
Prepared By:	ATLAS Computing Group

All trademarks, copyright names and products referred to in this document are acknowledged as such.

ISBN 92-9083-250-9

ATLAS Collaboration

Armenia
Yerevan Physics Institute, Yerevan

Australia
Research Centre for High Energy Physics, Melbourne University, Melbourne
University of Sydney, Sydney

Austria
Institut für Experimentalphysik der Leopold-Franzens-Universität Innsbruck, Innsbruck

Azerbaijan Republic
Institute of Physics, Azerbaijan Academy of Science, Baku

Republic of Belarus
Institute of Physics, National Academy of Science, Minsk
National Centre for Particle and High Energy Physics, Minsk

Brazil
Universidade Federal do Rio de Janeiro, COPPE/EE/IF, Rio de Janeiro

Canada
University of Alberta, Edmonton
University of Carleton, Ottawa
Group of Particle Physics, University of Montreal, Montreal
Department of Physics, University of Toronto, Toronto
Simon Fraser University, Burnaby, BC
TRIUMF, Vancouver
Department of Physics, University of British Columbia, Vancouver
University of Victoria, Victoria

CERN
European Organization for Nuclear Research (CERN), Geneva

China
Joint Cluster formed by IHEP Beijing, USTC Hefei, University of Nanjing and University of Shandong

Czech Republic
Academy of Sciences of the Czech Republic, Institute of Physics and Institute for Computer Science, Prague
Charles University in Prague, Faculty of Mathematics and Physics, Prague
Czech Technical University in Prague, Faculty of Nuclear Sciences and Physical Engineering, Faculty of Mechanical Engineering, Prague

Denmark
Niels Bohr Institute, University of Copenhagen, Copenhagen

France
Laboratoire d'Annecy-le-Vieux de Physique des Particules (LAPP), IN2P3-CNRS, Annecy-le-Vieux
Laboratoire de Physique Corpusculaire, Université Blaise Pascal, IN2P3-CNRS, Clermont-Ferrand
Laboratoire de Physique Subatomique et de Cosmologie de Grenoble, IN2P3-CNRS Université Joseph Fourier, Grenoble
Centre de Physique des Particules de Marseille, IN2P3-CNRS, Marseille
Laboratoire de l'Accélérateur Linéaire, IN2P3-CNRS, Orsay
LPNHE, Universités de Paris VI et VII, IN2P3-CNRS, Paris
Commissariat à l'Energie Atomique (CEA), DSM/DAPNIA, Centre d'Etudes de Saclay, Gif-sur-Yvette

Georgia
Institute of Physics of the Georgian Academy of Sciences and Tbilisi State University, Tbilisi

Germany
Physikalisches Institut, Universität Bonn, Bonn
Institut für Physik, Universität Dortmund, Dortmund
Fakultät für Physik, Albert-Ludwigs-Universität, Freiburg
Kirchhoff-Institut für Physik der Universität Heidelberg, Heidelberg
Institut für Physik, Universität Mainz, Mainz
Lehrstuhl für Informatik V, Universität Mannheim, Mannheim
Sektion Physik, Ludwig-Maximilian-Universität München, Munich
Max-Planck-Institut für Physik, Munich
Fachbereich Physik, Universität Siegen, Siegen
Fachbereich Physik, Bergische Universität, Wuppertal

Greece
Athens National Technical University, Athens
Athens University, Athens
Aristotle University of Thessaloniki, High Energy Physics Department and Department of Mechanical Engineering, Thessaloniki

Israel
Department of Physics, Technion, Haifa
Raymond and Beverly Sackler Faculty of Exact Sciences, School of Physics and Astronomy, Tel-Aviv University, Tel-Aviv
Department of Particle Physics, The Weizmann Institute of Science, Rehovot

Italy
Dipartimento di Fisica dell'Università della Calabria e I.N.F.N., Cosenza
Laboratori Nazionali di Frascati dell'I.N.F.N., Frascati
Dipartimento di Fisica dell'Università di Genova e I.N.F.N., Genova
Dipartimento di Fisica dell'Università di Lecce e I.N.F.N., Lecce
Dipartimento di Fisica dell'Università di Milano e I.N.F.N., Milan
Dipartimento di Scienze Fisiche, Università di Napoli 'Federico II' e I.N.F.N., Naples
Dipartimento di Fisica Nucleare e Teorica dell'Università di Pavia e I.N.F.N., Pavia
Dipartimento di Fisica dell'Università di Pisa e I.N.F.N., Pisa
Dipartimento di Fisica dell'Università di Roma I 'La Sapienza' e I.N.F.N., Roma
Dipartimento di Fisica dell'Università di Roma II 'Tor Vergata' e I.N.F.N., Roma
Dipartimento di Fisica dell'Università di Roma III 'Roma Tre' e I.N.F.N., Roma
Dipartimento di Fisica dell'Università di Udine, Gruppo collegato di Udine I.N.F.N. Trieste, Udine

Japan
Hiroshima Institute of Technology, Hiroshima
Department of Physics, Hiroshima University, Higashi-Hiroshima
KEK, High Energy Accelerator Research Organisation, Tsukuba
Department of Physics, Faculty of Science, Kobe University, Kobe
Department of Physics, Kyoto University, Kyoto
Kyoto University of Education, Kyoto
Nagasaki Institute of Applied Science, Nagasaki
Naruto University of Education, Naruto
Department of Physics, Faculty of Science, Okayama University, Okayama
Department of Computer Science, Ritsumeikan University, Kusatsu
Department of Physics, Faculty of Science, Shinshu University, Matsumoto
International Center for Elementary Particle Physics, University of Tokyo, Tokyo
Physics Department, Tokyo Metropolitan University, Tokyo
Department of Applied Physics System, Tokyo University of Agriculture and Technology, Tokyo
Institute of Physics, University of Tsukuba, Tsukuba

Morocco
Faculté des Sciences Aïn Chock, Université Hassan II, Casablanca, and Université Mohamed V, Rabat

Netherlands
FOM - Institute SAF NIKHEF and University of Amsterdam/NIKHEF
University of Nijmegen/NIKHEF, Nijmegen

Norway
University of Bergen, Bergen
University of Oslo, Oslo

Poland
Henryk Niewodniczanski Institute of Nuclear Physics, Cracow
Faculty of Physics and Nuclear Techniques of the University of Mining and Metallurgy, Cracow

Portugal
Laboratório de Instrumentação e Física Experimental de Partículas (LIP), Lisbon, in collaboration with: Faculdade de Ciências de Universidade de Lisboa (FCUL), University of Coimbra, University Católica-Figueira da Foz, and University Nova de Lisboa

Romania
National Institute for Physics and Nuclear Engineering, Institute of Atomic Physics, Bucharest

Russia
Institute for Theoretical and Experimental Physics (ITEP), Moscow
P.N. Lebedev Institute of Physics, Moscow
Moscow Engineering and Physics Institute (MEPhI), Moscow
Moscow State University, Moscow
Budker Institute of Nuclear Physics (BINP), Novosibirsk
State Research Center of the Russian Federation - Institute for High Energy Physics (IHEP), Protvino
Petersburg Nuclear Physics Institute (PNPI), Gatchina, St. Petersburg

JINR
Joint Institute for Nuclear Research , Dubna

Republic of Serbia
Institute of Physics, University of Belgrade, Belgrade

Slovak Republic
Bratislava University, Bratislava, and Institute of Experimental Physics of the Slovak Academy of Sciences, Kosice

Slovenia
Jozef Stefan Institute and Department of Physics, University of Ljubljana, Ljubljana

Spain
Institut de Física d'Altes Energies (IFAE), Universidad Autónoma de Barcelona, Bellaterra, Barcelona
Physics Department, Universidad Autónoma de Madrid, Madrid
Instituto de Física Corpuscular (IFIC), Centro Mixto Universidad de Valencia - CSIC, Valencia and Instituto de Microelectrónica de Barcelona, Bellaterra, Barcelona

Sweden
Fysika institutionen, Lunds universitet, Lund
Royal Institute of Technology (KTH), Stockholm
University of Stockholm, Stockholm
Uppsala University, Department of Radiation Sciences, Uppsala

Switzerland
Laboratory for High Energy Physics, University of Bern, Bern
Section de Physique, Université de Genève, Geneva

Taiwan
Institute of Physics, Academia Sinica, Taipei

Turkey
Department of Physics, Ankara University, Ankara
Department of Physics, Bogaziçi University, Istanbul

United Kingdom
School of Physics and Astronomy, The University of Birmingham, Birmingham
Cavendish Laboratory, Cambridge University, Cambridge
Department of Physics and Astronomy, University of Glasgow, Glasgow
Department of Physics, Lancaster University, Lancaster
Department of Physics, Oliver Lodge Laboratory, University of Liverpool, Liverpool
Department of Physics, Queen Mary and Westfield College, University of London, London
Department of Physics, Royal Holloway, University of London, Egham
Department of Physics and Astronomy, University College London, London
Department of Physics and Astronomy, University of Manchester, Manchester
Department of Physics, Oxford University, Oxford
Particle Physics Department, Rutherford Appleton Laboratory, Chilton, Didcot
Department of Physics, University of Sheffield, Sheffield

United States of America
State University of New York at Albany, New York
Argonne National Laboratory, Argonne, Illinois
University of Arizona, Tucson, Arizona
Department of Physics, The University of Texas at Arlington, Arlington, Texas
Lawrence Berkeley Laboratory and University of California, Berkeley, California
Physics Department of the University of Boston, Boston, Massachusetts
Brandeis University, Department of Physics, Waltham, Massachusetts
Brookhaven National Laboratory (BNL), Upton, New York
University of Chicago, Enrico Fermi Institute, Chicago, Illinois
Nevis Laboratory, Columbia University, Irvington, New York
Department of Physics, Duke University, Durham, North Carolina
Department of Physics, Hampton University, Virginia
Department of Physics, Harvard University, Cambridge, Massachusetts
Indiana University, Bloomington, Indiana
Iowa State University, Ames, Iowa
University of California, Irvine, California
Department of Physics, University of Massachusetts, Amherst
Massachusetts Institute of Technology, Department of Physics, Cambridge, Massachusetts
Michigan State University, Department of Physics and Astronomy, East Lansing, Michigan
University of Michigan, Department of Physics, Ann Arbor, Michigan
Department of Physics, New Mexico University, Albuquerque, New Mexico
Ohio State University, Columbus, Ohio
Department of Physics and Astronomy, University of Oklahoma
Department of Physics, University of Pennsylvania, Philadelphia, Pennsylvania
University of Pittsburgh, Pittsburgh, Pennsylvania
Department of Physics and Astronomy, University of Rochester, Rochester, New York
Institute for Particle Physics, University of California, Santa Cruz, California
Department of Physics, Southern Methodist University, Dallas, Texas
State University of New York at Stony Brook, New York
Tufts University, Medford, Massachusetts
High Energy Physics, University of Illinois, Urbana, Illinois
Department of Physics, Department of Mechanical Engineering, University of Washington, Seattle, Washington
Department of Physics, University of Wisconsin, Madison, Wisconsin
Yale University, New Haven, Connecticut

Acknowledgements

The editors would like to thank all members of the ATLAS Collaboration, especially the many who have contributed directly to this Computing Technical Design Report and those who provided feedback on draft versions of it.

The editors would also like to thank Susan Leech O'Neale for her invaluable contribution as proofreader and Stephane Viollet and the technical staff of the CERN Print Shop for their help.

Editorial Board

Günter Duckeck (chair)

Dario Barberis

Richard Hawkings

Roger Jones

Norman McCubbin

Gilbert Poulard

David Quarrie

Torre Wenaus

Emil Obreshkov (technical editor)

Contents

ATLAS Collaboration . . . . . **iii**

Acknowledgements . . . . . **viii**
Editorial Board viii

1 Introduction **1**
1.1 Overview 1
1.2 Outlook 3

2 Computing Model **5**
2.1 Introduction 5
2.2 Input Parameters for Proton-Proton Collisions 6
2.2.1 LHC Operational Parameters and Trigger Rates 6
2.2.2 Types of Data 6
2.2.3 Event Store 7
2.2.4 The Tier Structure and the Roles of the Various Tiers 8
2.2.5 Data Flow 11
2.3 Data Analysis 16
2.3.1 Analysis Procedures and Data Flow 16
2.3.2 The Analysis Model and Access to Resources 17
2.3.3 Distributed Analysis System 18
2.4 Simulation Process 18
2.5 Calibration and Alignment 19
2.5.1 Types of Processing 19
2.5.2 Calibration Streams 20
2.5.3 Prompt Reconstruction Latency 21
2.5.4 Offline Calibration and Alignment 21
2.6 Heavy Ion Data 22
2.7 Commissioning the System 23
2.8 Summary 24
2.9 References 24

3 Offline Software **25**
3.1 Introduction and Requirements 25
3.2 Methodology and Domain Decomposition 25
3.2.1 An Object-Oriented Methodology 25
3.2.2 Domain Decomposition 26
3.3 The Athena Framework 27
3.3.1 Introduction 27
3.4 Data Access Model 32
3.4.1 Data Objects and Algorithms 32
3.4.2 StoreGate: the ATLAS Transient Data Store 32
3.4.3 Data Objects 34
3.4.4 Accessing Data Objects 34
3.4.5 Using DataLinks to Persistify References 35

3.5 Detector Description . . . 35
3.5.1 ATLAS Geometry DB Schema - the Hierarchical Versioning System (HVS) . . . 36
3.5.2 The Geometry Kernel (GeoModel) . . . 37
3.5.3 Generic Identification Scheme Connecting Events and Detector Description . . . 41
3.6 Graphics and Event Display . . . 43
3.6.1 Introduction . . . 43
3.6.2 Atlantis . . . 44
3.6.3 HEPVis/v-atlas . . . 45
3.6.4 Persint . . . 47
3.7 Calibration and Alignment . . . 49
3.7.1 Inner Detector . . . 49
3.7.2 Liquid Argon and Tile Calorimeters . . . 51
3.7.3 Muon Spectrometer . . . 52
3.8 Simulation . . . 53
3.8.1 The Simulation Data Flow . . . 53
3.8.2 Generators. . . . 54
3.8.3 Fast Simulation (Atlfast) . . . 56
3.8.4 ATLAS Geant4 Simulation (G4ATLAS) . . . 57
3.8.5 Pile-up . . . 59
3.8.6 Digitization . . . 60
3.9 Event Selection and Reconstruction. . . . 60
3.9.1 Introduction . . . 60
3.9.2 Reconstruction . . . 61
3.9.3 Analysis Preparation . . . 67
3.10 Physics Analysis Tools . . . 70
3.10.1 Current Status . . . 71
3.10.2 Short-Term Objectives. . . . 72
3.11 Use of Offline Software for HLT and Monitoring . . . 73
3.12 Use of Offline Software for Detector Commissioning. . . . 74
3.13 Testing and Validation . . . 78
3.13.1 Testing Aims . . . 78
3.13.2 Types of Testing. . . . 78
3.13.3 Testing Frameworks . . . 79
3.13.4 Testing Future . . . 80
3.13.5 Physics Validation . . . 81
3.14 Software Infrastructure . . . 81
3.14.1 Introduction and Description . . . 81
3.14.2 Code Management . . . 82
3.14.3 External Packages . . . 84
3.14.4 Platforms and Compilers. . . . 84
3.14.5 Releases and Release Strategy . . . 85
3.14.6 Code Distribution . . . 87
3.14.7 Quality Assurance and Quality Control (QA/QC) . . . 87
3.14.8 Documentation . . . 88

3.14.9 User Support . . . 88
3.14.10 HLT Coordination . . . 89
3.14.11 Mailing Lists . . . 89
3.15 Integration of LCG Application Area Products . . . 90
3.15.1 SEAL. . . . 90
3.15.2 POOL . . . 91
3.15.3 PI . . . 91
3.15.4 Simulation Components . . . 91
3.15.5 SPI Components . . . 92
3.15.6 LCGCMT . . . 93
3.16 References . . . 93

4 Databases and Data Management . . . 97
4.1 Introduction . . . 97
4.2 Approach and Architecture . . . 97
4.3 Technical Data . . . 99
4.3.1 Technical Coordination Databases . . . 99
4.3.2 Detector Production Data. . . . 100
4.3.3 Installation Data . . . 100
4.3.4 Survey Database . . . 101
4.4 Conditions and Configuration Data . . . 101
4.4.1 Introduction . . . 101
4.4.2 Conditions Database Architecture. . . . 103
4.4.3 Conditions Database Online Model . . . 105
4.4.4 Supporting Tools . . . 107
4.4.5 Athena Access to Conditions Data . . . 107
4.4.6 Performance, Scalability and Distribution . . . 108
4.4.7 Detector Description . . . 109
4.5 Event Data . . . 110
4.5.1 Introduction . . . 110
4.5.2 Persistence Services. . . . 110
4.5.3 Storage Technology and Technology Selection . . . 112
4.5.4 Persistence of the Event Data Model . . . 112
4.5.5 Automatic Persistence . . . 112
4.5.6 Navigational Infrastructure . . . 113
4.5.7 Schema Evolution . . . 113
4.5.8 Event-level Metadata . . . 114
4.5.9 Event Selection and Input Specification . . . 114
4.5.10 Event Store Services . . . 115
4.5.11 Summary . . . 115
4.6 Distributed Data Management . . . 115
4.6.1 Introduction . . . 115
4.6.2 DC2 Experience . . . 116
4.6.3 Precepts and Requirements . . . 117
4.6.4 Scenarios and Use Cases . . . 119
4.6.5 System Architecture . . . 121

4.6.6 Implementation 128
4.6.7 Organization of System Development, Deployment and Operation . 130
4.6.8 DQ2 Development and Deployment Timeline 131
4.7 Book-keeping for Production and Analysis 131
4.7.1 Production Database 131
4.7.2 Offline Book-keeping 132
4.7.3 Provenance and Related Metadata 133
4.8 Database and Data Management Services 134
4.8.1 Distributed Database Services 134
4.8.2 Caching Tiers for Distributed Database Access 135
4.8.3 Operational Experience 136
4.8.4 Monitoring 136
4.8.5 Roles. 137
4.8.6 Database Authentication 137
4.9 Development Processes and Infrastructure 138
4.9.1 Milestones, Planning, Manpower 139
4.10 References 139

5 Grid Tools and Services 141
5.1 Introduction. 141
5.2 Relations with the LCG Project and with Grid Middleware Providers . . 141
5.3 Integration of Grid Tools 142
5.3.1 The ATLAS Virtual Organization 142
5.3.2 Data Management on the Grid 143
5.3.3 Job Submission to the Grid 144
5.4 The Distributed Production System. 146
5.4.1 Architecture 146
5.4.2 Job Transformations 147
5.4.3 Production Database 148
5.4.4 Supervisor. 149
5.4.5 Executors 149
5.4.6 Experience with the Production System 155
5.4.7 Evolution of the Production System 156
5.5 Distributed Analysis on the Grid 162
5.5.1 The ATLAS Distributed Analysis Model 163
5.5.2 Data Management 163
5.5.3 Workload Management 164
5.5.4 Prototyping Activities. 164
5.5.5 The ADA Project 166
5.6 Testing and Commissioning of the System 167
5.7 References 168

6 Computing Operations 171
6.1 Introduction. 171
6.2 Software Distribution and Deployment 171
6.2.1 Preparation and Distribution of ATLAS Software 171

6.2.2 ATLAS Software Deployment on the Grid . . . 171
6.3 Production Operations . . . 173
6.3.1 Operations at Tier-0 . . . 173
6.3.2 Operations at Tier-1s . . . 180
6.3.3 Operations at Tier-2s . . . 181
6.3.4 Production and Databases . . . 182
6.3.5 Monitoring and Book-Keeping . . . 183
6.4 Productions . . . 183
6.4.1 Distributed Production and Analysis Operations . . . 184
6.4.2 Management of the ATLAS Virtual Organization . . . 184
6.5 Experience with Data Challenges and other Mass Productions . . . 186
6.5.1 Data Challenge 1. . . . 187
6.5.2 Data Challenge 2. . . . 187
6.5.3 Production for the ATLAS Physics Workshop . . . 190
6.5.4 Combined Test Beam . . . 192
6.6 Summary . . . 194
6.7 References . . . 194

7 Resource Requirements . . . 197
7.1 Introduction . . . 197
7.2 Ramp-up and Resource Requirement Evolution . . . 198
7.3 Networking Requirements . . . 201

8 Project Organization and Planning . . . 205
8.1 Organizational Structure . . . 205
8.1.1 The Computing Management Board . . . 206
8.1.2 The Computing Oversight Board . . . 206
8.1.3 The International Computing Board . . . 207
8.1.4 The Software Project . . . 207
8.1.5 The Database and Data Management Project . . . 209
8.1.6 Grid Tools and Services . . . 209
8.1.7 Computing Operations . . . 210
8.1.8 Computing Resources Management . . . 210
8.1.9 Relations with other ATLAS-wide Projects. . . . 210
8.1.10 Relations with the LCG Project . . . 212
8.2 Work Plan, Schedule and Milestones . . . 212
8.2.1 Planning Methodology . . . 212
8.2.2 Data Challenge 3: Computing System Commissioning . . . 213
8.2.3 High-Level Milestones. . . . 214
8.3 Manpower and Hardware Resources . . . 215
8.3.1 Manpower Resources . . . 215
8.3.2 Hardware Resources . . . 218
8.4 References . . . 219

A Glossary . . . 221
A.1 Acronyms and Terms . . . 221
A.2 Definitions . . . 225

1 Introduction

This Technical Design Report describes the ATLAS Software & Computing Project. The project covers all developments of offline software for the ATLAS experiment, the definition of the computing environment, the provision of hardware and manpower resources, and the operation of the ATLAS offline Computing system.

In particular, the Software & Computing Project is responsible for the provision of the software framework and services, the data management system, user-support services, and the world-wide data access and job-submission system. The development of detector-specific algorithmic code for simulation, calibration, alignment, trigger and reconstruction is under the responsibility of the detector projects, but the Software & Computing Project plans and coordinates these activities across detector boundaries. In particular, a significant effort has been made to ensure that relevant parts of the "offline" framework and event-reconstruction code can be used in the High-Level Trigger. Similarly, close cooperation with Physics Coordination and the Combined Performance groups ensures the smooth development of global event-reconstruction code and of software tools for physics analysis.

The ATLAS Computing Model is described in Chapter 2. The high-level Computing Model defines the requirements on the architecture and performance of the software and of the overall computing system. Chapter 3 describes all software development activities that are part of the Software & Computing Project. The ATLAS Collaboration set up in 2004 an ATLAS-wide Database and Data Management Project, which is responsible for all data management issues, from technical detector construction data to online run configuration and world-wide distribution of event and conditions data; this project is described in Chapter 4.

The needs of large-scale simulation, reconstruction and analysis of ATLAS data require the development of a world-wide distributed computing system, which allows efficient data access and makes use of all available computing resources; the developments towards such a system are described in Chapter 5. Chapter 6 describes the experience we have had so far with large-scale distributed computing operations, in the context of the Data Challenges that were run from 2001 onwards, and the development towards the operation system that will be needed at experiment turn-on. Taking into account the Computing Model and the current, and extrapolated, performance of ATLAS software and other computing tools, Chapter 7 lists the hardware resources that will be required in order to process and analyse ATLAS data. Finally, the organization of the Software & Computing Project is described in Chapter 8, together with a summary of high-level project milestones.

1.1 Overview

The ATLAS Computing Model embraces the Grid paradigm and a high degree of decentralization and sharing of computing resources. The required level of computing resources means that off-site facilities will be vital to the operation of ATLAS in a way that was not the case for previous CERN-based experiments.

The primary event processing occurs at CERN in a Tier-0 facility. The RAW data is archived at CERN and copied (along with the primary processed data) to the Tier-1 facilities around the world. These facilities archive the raw data, provide the reprocessing capacity, provide access to the various processed versions, and allow scheduled analysis of the processed data by physics

analysis groups. Derived datasets produced by the physics groups are copied to the Tier-2 facilities for further analysis. The Tier-2 facilities also provide the simulation capacity for the experiment, with the simulated data housed at Tier-1s. In addition, Tier-2 centres will provide analysis facilities, and some will provide the capacity to produce calibrations based on processing raw data. A CERN Analysis Facility provides an additional analysis capacity, with an important role in the calibration and algorithmic development work.

ATLAS has adopted an object-oriented approach to software, based primarily on the C++ programming language, but with some components implemented using FORTRAN and Java. A component-based model has been adopted, whereby applications are built up from collections of plug-compatible components based on a variety of configuration files. This capability is supported by a common framework that provides common data-processing support. This approach results in great flexibility in meeting the basic processing needs of the experiment, and also for responding to changing requirements throughout its lifetime. The heavy use of abstract interfaces allows for different implementations to be provided, supporting different persistency technologies, or optimized for the offline or high-level trigger environments.

The Athena framework is an enhanced version of the Gaudi framework that was originally developed by the LHCb experiment, but is now a common ATLAS–LHCb project. Major design principles are the clear separation of data and algorithms, and of transient (in-memory) and persistent (in-file) data. All levels of processing of ATLAS data, from high-level trigger to event simulation, reconstruction and analysis, take place within the Athena framework; in this way it is easier for code developers and users to test and run algorithmic code, with the assurance that all geometry and conditions data will be the same for all types of applications (simulation, reconstruction, analysis, visualization).

One of the principal challenges for ATLAS computing is to develop and operate a data storage and management infrastructure able to meet the demands of a yearly data volume of O(10 PB) utilized by data processing and analysis activities spread around the world. The ATLAS Computing Model establishes the environment and operational requirements that ATLAS data-handling systems must support, and, together with the operational experience gained to date in test beams and data challenges, provides the primary guidance for the development of the data management systems.

The ATLAS Databases and Data Management Project (DB Project) leads and coordinates ATLAS activities in these areas, with a scope encompassing technical databases (detector production, installation and survey data), detector geometry, online/TDAQ databases, conditions databases (online and offline), event data, offline processing configuration and book-keeping, distributed data management, and distributed database and data management services. The project is responsible for ensuring the coherent development, integration, and operational capability of the distributed database and data management software and infrastructure for ATLAS across these areas.

The ATLAS Computing Model foresees the distribution of raw and processed data to Tier-1 and Tier-2 centres, so as to be able to exploit fully the computing resources that are made available to the Collaboration. Additional computing resources will be available for data processing and analysis at Tier-3 centres and other computing facilities to which ATLAS may have access. A complex set of tools and distributed services, enabling the automatic distribution and processing of the large amounts of data, has been developed and deployed by ATLAS in cooperation with the LHC Computing Grid (LCG) Project and with the middleware providers of the three large Grid infrastructures we use: EGEE, OSG and NorduGrid. The tools are designed in a flexible way, in order to have the possibility to extend them to use other types of Grid middleware

in the future. These tools, and the service infrastructure on which they depend, were initially developed in the context of centrally managed, distributed Monte Carlo production exercises. They will be re-used wherever possible to create systems and tools for individual users to access data and compute resources, providing a distributed analysis environment for general usage by the ATLAS Collaboration. The first version of the production system was deployed in summer 2004 and has been used since the second half of 2004. It was used for Data Challenge 2, for the production of simulated data for the 5th ATLAS Physics Workshop (Rome, June 2005) and for the reconstruction and analysis of the 2004 Combined Test-Beam data.

The main computing operations that ATLAS will have to run comprise the preparation, distribution and validation of ATLAS software, and the computing and data management operations run centrally on Tier-0, Tier-1s and Tier-2s. The ATLAS Virtual Organization will allow production and analysis users to run jobs and access data at remote sites using the ATLAS-developed Grid tools.

In the past few years the Computing Model has been tested and developed by running Data Challenges of increasing scope and magnitude, as was proposed by the LHC Computing Review in 2001. We have run two major Data Challenges since 2002 and performed other massive productions in order to provide simulated data to the physicists and to reconstruct and analyse real data coming from test-beam activities; this experience is now useful in setting up the operations model for the start of LHC data-taking in 2007.

The Computing Model, together with the knowledge of the resources needed to store and process each ATLAS event, gives rise to estimates of required resources that can be used to design and set up the various facilities. It is not assumed that all Tier-1s or Tier-2s will be of the same size; however, in order to ensure a smooth operation of the Computing Model, all Tier-1s should have broadly similar proportions of disk, tape and CPU, and the same should apply for the Tier-2s.

The organization of the ATLAS Software & Computing Project reflects all areas of activity within the project itself. Strong high-level links have been established with other parts of the ATLAS organization, such as the T-DAQ Project and Physics Coordination, through cross-representation in the respective steering boards. The Computing Management Board, and in particular the Planning Officer, acts to make sure that software and computing developments take place coherently across sub-systems and that the project as a whole meets its milestones. The International Computing Board assures the information flow between the ATLAS Software & Computing Project and the national resources and their Funding Agencies.

1.2 Outlook

As described in detail in the rest of this document, the ATLAS Software & Computing Project has made major progress towards the realization of its goals, although significant development is still required before LHC turn-on. We are confident that we can be ready to meet the challenge of first data, and that our overall strategy is sufficiently flexible to respond rapidly and effectively to the surprises that real data will undoubtedly bring.

Of course we count on a continuing, fruitful collaboration with all the non-ATLAS projects working towards LHC computing, particularly the LCG and other Grid-oriented projects. Within ATLAS we have identified a requirement for a small amount of additional effort in critical ar-

eas, notably in database technology and other areas of computing infrastructure; the provision of this effort would be beneficial to a degree that far outweighs its cost.

2 Computing Model

2.1 Introduction

The primary purpose of this chapter is to present the current ideas on the steady-state ATLAS offline Computing Model and to detail the tests performed to validate that model. The current best estimate of the resources implied is given in Chapter 7. The model extends from the primary event store, which is where events selected by the trigger are recorded, to the analyst at a remote university. Consideration is also given to computing issues in the DAQ and the High Level Trigger (HLT) farm, that is prior to the primary event store.

Also considered is the model for the commissioning of the computing system with real data. This will require enhanced access to raw and nearly-raw data for calibration, algorithm development etc, with resultant implications for the resource providers.

The ideas and estimates presented here have evolved from previous studies, including the ATLAS computing resource and cost estimates developed using the MONARC hierarchical model [2-1] for the CERN LHC Computing Review [2-2]. The advent of the World Wide Grid triggered new ideas on how to organize the ATLAS system as a virtual worldwide distributed computing facility. This was presented to a subsequent CERN LHC Computing Review [2-3].

The main requirements on the Computing Model are to enable all members of the ATLAS Collaboration speedy access to all reconstructed data for analysis during the data-taking period, and appropriate access to raw data for organized monitoring, calibration and alignment activities. The model presented here makes substantial use of Grid Computing concepts, thereby allowing the same level of data access, and making available the same amount of computing resources, to all members of the ATLAS Collaboration.

The Computing Model embraces the Grid paradigm and a high degree of decentralization and sharing of computing resources. However, as different computer facilities are better suited to different roles, a degree of hierarchy, with distinct roles at each level, remains. This should not obscure the fact that all of the roles described are vital and must receive due weight. The required level of computing resources means that off-site facilities will be vital to the operation of ATLAS in a way that was not the case for previous CERN-based experiments.

The primary event processing occurs at CERN in a Tier-0 Facility. The RAW data is archived at CERN and copied (along with the primary processed data) to the Tier-1 facilities around the world. These facilities archive the RAW data, provide the reprocessing capacity, provide access to the various processed versions and allow scheduled analysis of the processed data by physics analysis groups. Derived datasets produced by the physics groups are copied to the Tier-2 facilities for further analysis. The Tier-2 facilities also provide the simulation capacity for the experiment, with the simulated data housed at Tier-1s. In addition, Tier-2 centres will provide analysis facilities and some will provide the capacity to produce calibrations based on processing some raw data. A CERN Analysis Facility provides an additional analysis capacity, with an important role in the data-intensive calibration and algorithmic development work.

ATLAS will negotiate relationships between Tier-1s and Tier-2s, and also among Tier-1s themselves, to try to optimize the smooth running of the system in terms of data transfer, balanced storage and network topologies. It is not assumed that all Tier-1s or Tier-2s will be of the same size. However, the ratio of disk, tape and CPU resources required is the same in each case.

2.2 Input Parameters for Proton-Proton Collisions

Input parameters for the offline Computing Model are derived from the information contained in the "HLT/DAQ TDR" [2-4] and the "Physics TDR" [2-5]. All input parameters are to be considered as reference numbers representing our best estimates at the moment. In the following sections, the ATLAS Computing Model is first worked out for one year of steady-state operation of proton-proton data taking. Later chapters deal with the commissioning of the system and the resource requirements as a function of time.

2.2.1 LHC Operational Parameters and Trigger Rates

Table 2-1 The assumed LHC operational parameters and ATLAS trigger rates

E	14 TeV (two 7 TeV proton beams)
L	$0.5*10^{33}$ $cm^{-2}s^{-1}$ in 2007
	$2*10^{33}$ $cm^{-2}s^{-1}$ in 2008 and 2009
	10^{34} $cm^{-2}s^{-1}$ (design luminosity) from 2010 onwards
σ	100 mb = 10^{-25} cm^2
Collision rate	$L \cdot \sigma = 10^9$ Hz p-p collisions at design luminosity
Trigger rate	200 Hz independent of the luminosity

The machine and trigger parameters assumed for the Computing Model are summarized in Table 2-1. It is assumed that the trigger thresholds and selection conditions will be adjusted continually so as to maximize the physics reach of the experiment.

2.2.2 Types of Data

The assumptions of Table 2-2 are used to calculate the storage and computing resources (the definitions of the various data types are given in the next section).

The assumed processing times are projections based on those for the current code, in the light of planned future improvements and known inefficiencies, and are for the running conditions in 2008 and 2009. Between data without pile-up and the pile-up for the design luminosity of 10^{34} $cm^{-2}s^{-1}$, the event sizes are seen to grow by 50% and the processing time by 75%; this information is used in the resource evolution projections. At present, all processing-time numbers are higher than assumed here, the reconstruction by a factor of two and the simulation by a factor of about four. For the event sizes, the first prototype of the AOD is about 60% that assumed here (but has yet to be tested in terms of the required functionality for analysis). The other data formats are presently larger than the target size, but the target sizes are believed to be achievable. (For example, the RAW data size is 1.7 MB without pile-up, but twice this size after pile-up at full luminosity.)

Table 2-2 The assumed event data sizes for various formats, the corresponding processing times and related operational parameters.

Item	Unit	Value
Raw Data Size	MB	1.6
ESD Size	MB	0.5
AOD Size	kB	100
TAG Size	kB	1
Simulated Data Size	MB	2.0
Simulated ESD Size	MB	0.5
Time for Reconstruction (1 ev)	kSI2k-sec	15
Time for Simulation (1 ev)	kSI2k-sec	100
Time for Analysis (1 ev)	kSI2k-sec	0.5
Event rate after EF	Hz	200
Operation time	seconds/day	50000
Operation time	days/year	200
Operation time (2007)	days/year	50
Event statistics	events/day	10^7
Event statistics (from 2008 onwards)	events/year	$2 \cdot 10^9$

2.2.3 Event Store

The physics event store holds a number of successively derived event representations, beginning with raw or simulated data and progressing through reconstruction into more streamlined event representations suitable for analysis. Constituent components are described in the following paragraphs.

RAW Data: RAW data are events as output by the Event Filter (EF, the final stage of the HLT) for reconstruction. The model assumes an event size of 1.6 MB, arriving at an output rate of 200 Hz (including 20 Hz of calibration trigger data).

Events arrive from the Event Filter in “byte-stream” format, reflecting the format in which data are delivered from the detector, rather than in any object-oriented representation. Events will be transferred from the EF to the Tier-0 in files of at most 2 GB. Each file will contain events belonging to a single run (corresponding to a prolonged period of data taking using the same trigger selections on the same fill in the accelerator), but the events in each file will not be consecutive nor ordered [2-6].

Event Summary Data (ESD): ESD refers to event data written as the output of the reconstruction process. ESD is intermediate in size between RAW and Analysis Object Data (see below). Its content is intended to make access to RAW data unnecessary for most physics applications other than for some calibration or re-reconstruction. ESD has an object-oriented representation, and is stored in POOL ROOT files. The target size is 500 kB per event [2-7].

Analysis Object Data (AOD): AOD is a reduced event representation, derived from ESD, suitable for analysis. It contains physics objects and other elements of analysis interest. The target size is 100 kB per event. It has an object-oriented representation, and is stored in POOL ROOT files.

Tag Data (TAG): TAG data are event-level metadata — thumbnail information about events to support efficient identification and selection of events of interest to a given analysis. To facilitate queries for event selection, TAG data are stored in a relational database. The assumed average size is 1 kB per event.

Derived Physics Data (DPD): DPD is an n-tuple-style representation of event data for end-user analysis and histogramming. The inclusion of DPD in the Computing Model is an acknowledgment of the common practice by physicists of building subsamples in a format suitable for direct analysis and display by means of standard analysis tools (PAW, ROOT, JAS etc.), though software providers certainly expect that analysis, histogramming, and display via standard tools will be possible with AOD as input.

Simulated Event Data (SIM): SIM refers to a range of data types, beginning with generator events (e.g. from Pythia or similar programs) through simulation of interactions with the detector (e.g. Geant4 hits) and of detector response (digitization). It may also include pile-up, with the superposition of minimum bias events, or the simulation of cavern background. Events may be stored after any of these processing stages. The storage technology of choice is POOL ROOT files. Digitized events may alternatively be stored in byte-stream format for trigger studies or for emulation of data coming from the Event Filter. Simulated events are often somewhat larger than RAW events (approximately 2 MB in size), in part because they usually retain Monte Carlo "truth" information.

Other formats are allowed in the software and processing model that are not included in the baseline. For example, the Derived Reconstruction Data (DRD) is an option being considered for the early phase of data taking for a subset of the data. It consists of raw data augmented with partially reconstructed objects to allow easy calibration and optimization of detector code. This format is only of use if the trade-off between storage cost and CPU to derive the partial-reconstruction is in the favour of storage. This in turn depends on the sample size required and the number of times the sample is passed-over by the detector groups.

2.2.4 The Tier Structure and the Roles of the Various Tiers

While the ATLAS Computing Model is very much Grid-based, there still remain distinct roles for different facilities that may be characterized in the following ways. It is to be stressed that all are important and make an invaluable contribution to ATLAS, and a sensible balance between the resources in the various Tiers is essential for the operation of the Computing Model. A guiding principle is to ensure two backed-up copies of the main data formats are made, giving security against failures.

2.2.4.1 Tier-0 at CERN

The Tier-0 facility at CERN is responsible for the archiving and distribution of the primary RAW data received from the Event Filter. It provides the prompt reconstruction of the calibration and express streams and the somewhat slower first-pass processing of the primary event stream. The derived datasets (ESD, primary AOD and TAG sets) are distributed from the Tier-0 to the

Tier-1 facilities described below, and the reconstructed calibration data to the CERN Analysis Facility for data-intensive calibration. More automated calibration tasks will also be run by the Tier-0.

The Tier-0 must provide an extremely high availability and response time in the case of errors. In the event of prolonged down-time, first-pass processing and calibration must be taken over by the Tier-1 facilities described below. To account for failures and network outages, a disk buffer corresponding to about 5 days of data production will be required for the data flowing into the Tier-0. A smaller output buffer will be required in case of failures in the transfer of the derived datasets offsite (although switching to an alternate Tier-1 destination must be possible in the system).

Access to the Tier-0 facility is granted only to people in the central production group and those providing the first-pass calibration.

2.2.4.2 Tier-1 Facilities

Approximately 10 Tier-1 facilities are planned world-wide that will serve ATLAS. They take responsibility to host and provide long-term access and archiving of a subset of the RAW data (on average 1/10th each). They also undertake to provide the capacity to perform the reprocessing of the RAW data under their curation, and to provide ATLAS-wide access to the derived ESD, AOD and TAG datasets, with the most up-to-date version of the data available with short latency ('on disk') and the previous version available but perhaps with a longer latency ('on tape'). The Tier-1s also undertake to host a secondary low-latency copy of the current ESD, AOD and TAG samples from another Tier-1, and the simulated data samples from Tier-2 facilities to improve access and provide fail-over. All of the datasets hosted are considered to be for the collaboration as a whole, and the storage and CPU pledged to be funded by the Tier-1 for that purpose.

The Tier-1s must allow access to and provide capacity to analyse all of the hosted samples, and will provide part of the calibration processing capacity. Modest RAW data samples must be available at short latency to allow calibration and algorithmic development. They will also host some of the physics-working-group DPD samples.

Tier-1 facilities are expected to have a high level of service in terms of availability and response time. Given the vital role in receiving the raw data and reprocessing, down-times in excess of 12 hours become problematic in terms of catching up with processing and with the storage elsewhere of RAW data. The fact that the ESD will be copied to two sites (see Section 2.2.5) reduces somewhat the reliance on a given Tier-1 for short periods.

Access to the Tier-1 facilities is essentially restricted to the production managers of the working groups and to the central production group for reprocessing.

2.2.4.3 Tier-2 Facilities

Tier-2 facilities may take a range of significant roles in ATLAS such as providing calibration constants, simulation and analysis. This range of roles will result in different sizes of the facilities. Tier-2 facilities also provide analysis capacity for physics working groups and subgroups. This analysis activity is generally chaotic in nature. They typically will host one third of the available current primary AOD and the full TAG samples. They will also host some of the physics group DPD samples, most likely in accordance with local interest. In addition, they will pro-

vide all of the required simulation capacity for the experiment (but with the simulated data typically migrated to the Tier-1 unless general on-demand access can be ensured at the site). Agreements on the primary host for the data from a given Tier-2 will be negotiated, although some flexibility will be required in the case of access problems. The relationships formed will be influenced by the ATLAS organizational plans and by the networking topology available. The primary host arrangement will help the planning of network links and may well follow the arrangements within a region for Grid operations and user support.

The Tier-2s will also host modest samples of RAW and ESD data for code development. Some Tier-2s may take significant role in calibration following the local detector interests and involvements. It is assumed that the typical users do not require back-navigation from AOD to ESD and beyond in most cases. When they do, it is further assumed that data-placement tools move the required (small) sample of ESD events to the same storage location as the AOD events. (Similar considerations apply if RAW data access is required.) In the case that larger samples need to be processed, it is planned that this would be done on the Tier-1 facilities by someone with production rights and with the agreement of the appropriate working group. The CERN Tier-1 Analysis Facility (see below) is also intended for tasks requiring larger than normal access to the ESD and RAW, and would be used within quotas by individual working-group members.

The level of service in terms of availability and response time expected of a Tier-2 is lower than for a Tier-1 (unless it chooses to host the simulated data it generates).

In principle, all members of the ATLAS virtual organization have access to a given Tier-2. In practice (and for operational optimization), heightened access to CPU and resources may be given to specific working groups at a particular site, according to a local policy agreed with the ATLAS central administration in a way that the ATLAS global policy is enforced over the aggregate of all sites. An example may be that DPD for the Higgs working group may be replicated to a subset of Tier-2 facilities, and the working-group members have heightened access to those facilities.

2.2.4.4 CERN Analysis Facility

The CERN analysis facility is, as the name suggests, primarily devoted to analysis, supporting a relatively large user community. It will also provide an important platform for calibration and code development. It will be particularly useful for user access to RAW data, given its co-location with the Tier-0 facility.

The CERN analysis facility is expected to have a level of service comparable to the Tier-0, given its key role in calibration and alignment and the requirement that these activities introduce minimal latency in the first-pass data processing. It is intended to be available to all members of the ATLAS virtual organization, but with priority to people with well defined roles in algorithmic development, calibration and alignment.

2.2.4.5 Tier-3 Resources

There will be a continuing need for local resources within an institution to store user ntuple-equivalents and allow work to proceed off the Grid. Clearly, the user expectations will grow for these facilities, and a site would already provide typically terabytes of storage for local use. Such 'Tier-3' facilities (which may be collections of desktops machines or local institute clusters) should be Grid-enabled, both to allow job submission and retrieval from the Grid, and to permit

resources to be used temporarily and with agreement as part of the Tier-2 activities. Such resources may be useful for simulation or for the collective analysis of datasets shared with a working group for some of the time.

The size of Tier-3 resources will depend on the local user community size and other factors, such as any specific software development or analysis activity foreseen in a given institute, and are therefore neither centrally planned nor controlled. It is nevertheless assumed that every active user will need O(1 TB) of local disk storage and a few kSI2k of CPU capacity to efficiently analyse ATLAS data.

2.2.5 Data Flow

The source of the input real data for the Computing Model is primarily the Event Filter (EF). Data passing directly from the online to offsite facilities for monitoring and calibration purposes will be discussed only briefly, as they have little impact on the total resources required, and are being studied. While the possibility of other locations for part of the EF is retained, the baseline assumption is that the EF resides at the ATLAS pit. Other arrangements have little impact on the Computing Model except on the network requirements from the ATLAS pit area. The input data to the EF will require approximately 10x10 Gb/s links with very high reliability. The output data requires an average 320 MB/s (3 Gb/s) link connecting it to the first-pass processing facility. Remote event filtering would require upwards of 10 Gb/s to the remote site, the precise bandwidth depending on the fraction of the Event Filter load migrated away from the ATLAS pit.

While the option of 'streaming data' at the EF should be retained, the baseline model assumes a single primary stream containing all physics events flowing from the Event Filter to Tier-0. Several other auxiliary streams are also planned, the most important of which is a calibration hotline containing calibration trigger events (which would most likely include certain physics event classes). This stream is required to produce calibrations of sufficient quality to allow a useful first-pass processing of the main stream with minimum latency. A working target (which remains to be shown to be achievable) is to process 50% of the data within 8 hours and 90% within 24 hours.

Two other auxiliary streams are planned. The first is an express-line of physics triggers containing about 5% of the full data rate. These will allow both the tuning of physics and detector algorithms and also a rapid alert on some high-profile physics triggers. It is to be stressed that any physics based on this stream must be validated with the 'standard' versions of the events in the primary physics stream. However, such a hot-line should lead to improved reconstruction. It is intended to make much of the early raw-data access in the model point to this and the calibration streams. The fractional rate of the express stream will vary with time, and will be discussed in the context of the commissioning.

The other auxiliary stream contains pathological events, for instance those that fail in the event filter. These may pass the standard Tier-0 processing, but if not they will attract the attention of the development team. They will be strongly rate-limited.

The following assumptions are made about output from the Event Filter and input to first-pass reconstruction:

- Event Filter processors send their outputs to one of 30-50 Sub-Farm Output managers (SFOs).

- Events are written to files in byte-stream format by SFOs.
- SFOs are equivalent to one another, and do not sort physics events by trigger or type.
- In normal operation, SFOs will fill a file to a specified size or event count threshold, then close the file and open a new one.
- Files are the unit of transfer from the Event Filter to the Tier-0 centre.
- Files are eligible for transfer as soon as they are filled and closed.
- The data acquisition system assigns run numbers and event numbers, and provides event time stamps.
- At run boundaries, files written by SFOs are closed and transferred to the Tier-0 centre: no RAW data file from the Event Filter contains events from more than one run.

Note that this process does not ensure that the events within a file are time ordered. The following steps occur when raw data arrives at the input-disk buffer of the first-pass processing facility (henceforth known as Tier-0):

1. the raw data file is copied to the CERN Mass Storage System;
2. the raw data file is copied to permanent mass storage in one of the Tier-1s;
3. calibration and alignment procedures are run on the corresponding calibration stream events;
4. the express stream is reconstructed with the best-estimate calibrations available;
5. once appropriate calibrations are in place, first-pass reconstruction ('prompt' reconstruction) is run on the primary event stream (containing all physics triggers), and the derived sets archived into the CERN mass storage system (these are known as the 'primary' data sets, subsequent reprocessing giving rise to better versions that supersede them);
6. two instances of the derived ESD are exported to external Tier-1 facilities; each Tier-1 site assumes principal responsibility for its fraction of such data, and retains a replica of another equal fraction of the ESD for which another Tier-1 site is principally responsible. Tier-1 sites make current ESD available on disk.[1] ESD distribution from CERN occurs at completion of first-pass reconstruction processing of each file. As physics applications may need to navigate from ESD to RAW data, it is convenient to use the same placement rules for ESD as for RAW, i.e., if a site hosts specific RAW events, then it also hosts the corresponding ESD. The proposed "one file in, one file out" model for ESD production jobs makes achieving such correspondence simpler.
7. the derived AOD is archived via the CERN analysis facility and an instance is shipped to each of the external Tier-1s (a full copy at each Tier-1);
8. the AOD copy at each Tier-1 is replicated and shared between the associated Tier-2 facilities;
9. the derived TAG is archived via the CERN analysis facility and an instance is copied to each Tier-1. These copies are then replicated to each Tier-2 in full.

Step 2, the transfer of the RAW data to external Tier-1 facilities, is an important requirement. These sites are the primary data sources for any later re-reconstruction of that data, and serve as

1. At least one Tier-1 site proposes to host the entire ESD. This is not precluded, but the site would nonetheless, like every other Tier-1, assume principal responsibility for its agreed fraction of the ESD.

the principal sources of CPU resources for any such reprocessing. It not only allows reprocessing of data, asynchronous with data taking, it also allows additional capacity to be employed if there is a backlog of first-pass processing at the Tier-0. Note that this implies a degree of control over the Tier-1 environment and processing that is comparable to that at the Tier-0.

Selected ESD will also be copied to Tier-2 sites for specialized purposes. Resource estimates reflect this fact, but the models and policies by which this replication may be accomplished are negotiated among Tier-1 centres and their associated Tier-2 sites.

The AOD and TAG distribution models are similar, but employ different replication infrastructure because TAG data are database-resident. AOD and TAG distribution from CERN occur upon completion of first-pass reconstruction processing of each run.

2.2.5.1 Rates, Latency, and Buffering

An output rate of 200 Hz is approximately 4 Hz per SFO. At 1.6 MB per event, each SFO will fill a 2GB file with approximately 1250 events every 5 minutes. This in itself sets a minimum time before processing can proceed for any stream. However, this merely sets the latency for the 'prompt' reconstruction; for the primary stream, the latencies will be set by the time until calibration, alignment, and other conditions data are available to Tier-0 processors as discussed in Section 2.5.3. Data arrives at the Tier-0 at a rate of 320 MB/s. A disk pool requires approximately 25 TB for each day of buffer capacity it is proposed to provide. Proposed criteria for overall production latency are:

- The express and calibration data streams reconstructed with less than 8 hours' latency;
- 90% of primary data stream reconstructed within 48 hours, the bulk beginning after approximately 24 hours.

The system is assumed to have an input disk buffer of 127 TB, which corresponds to approximately five days of data taking.

The processing of the calibration and alignment data, which crucially sets the latency for the bulk processing, is discussed in Section 2.5.

It should be noted that the first-pass processing provides the opportunity for more sophisticated filtering and compression/data reduction of the RAW data sample. As confidence is gained with the processing chain and the understanding of the detector, this may be taken advantage of to reduce the RAW data stored at the remote sites (and consequently the volume of derived data). However, for the baseline model we do not assume such a reduction. In the baseline model, the RAW data distributed to the Tier-1s is a straight copy of the data received from the Event Filter, and will often be shipped offsite before the first-pass processing has occurred.

2.2.5.2 First-pass ESD Production

First-pass ESD production takes place at the Tier-0 centre. The unit of ESD production is the run, as defined by the ATLAS data acquisition system. ESD production begins as soon as RAW data files and appropriate calibration and conditions data arrive at the Tier-0 centre. The Tier-0 centre provides processing resources sufficient to reconstruct events at the rate at which they arrive from the Event Filter. These resources are dedicated to ATLAS event reconstruction during periods of data taking. The current estimate of the CPU required is 3000 kSI2k (approximately 15 kSI2k-seconds per event times 200 events/second). It is assumed that in normal operations

all first-pass processing is conducted on the CERN Tier-0 facility, although the Tier-1 facilities could provide additional capacity in exceptional circumstances.

A new job is launched for each RAW data file arriving at the Tier-0 centre. Each ESD production job takes a single RAW event data file in byte-stream format as input and produces a single file of reconstructed events in a POOL ROOT file as output. With the current projection of 500 kB per event in the Event Summary Data (ESD), a 2 GB input file of 1250 1.6 MB RAW events yields a 625 MB output file of reconstructed (ESD) events as output.

2.2.5.3 First-pass AOD Production

Production of Analysis Object Data (AOD) from ESD is a lightweight process in terms of CPU resources, extracting and deriving physics information from the bulk output of reconstruction (ESD) for use in analysis. An AOD production job in principle takes one or more ESD files in POOL ROOT format as input, and produces one or more POOL ROOT files containing AOD as output. Current estimates suggest an AOD size of 100 kB per event.

AOD must be derivable from ESD without reference to RAW data, but one might imagine concatenating RAW+ESD+AOD production into a single job for the sake of efficiency. While the database and control framework infrastructure support such concatenation, the current model separates ESD from AOD production. The reason is that job concatenation results in a large number of small files: a concatenated job that takes a single 2 GB RAW event file as input, while producing a 625 MB ESD file, would produce only a 125 MB AOD file, even if only one output AOD stream is written. If AOD output is written to multiple streams, AOD files are likely to average approximately 12 MB in size. The proposed model for first-pass AOD production is therefore to run it as a separate step at the Tier-0 centre, using on the order of 50 ESD files as input to each AOD production job.

As AOD events will be read many times more often than ESD and RAW data, AOD events are physically clustered on output by trigger or physics channel or other criteria that reflect analysis access patterns. This means that an AOD production job, unlike an ESD production job, produces many output files. The baseline streaming model is that each AOD event is written to exactly one stream: AOD output streams comprise a disjoint partition of the run. All streams produced in first-pass reconstruction share the same definition of AOD. On the order of 10 streams are anticipated in first-pass reconstruction.

It is of course true that some events are of interest to more than one physics working group. Such events are nonetheless written only once, to avoid complications for analyses that cross stream boundaries (e.g., “Have I already seen this event in another stream?”). Streams should be thought of as heuristically-based data access optimizations: the idea is to try to reduce the number of files that need to be touched in an average analysis, not to produce perfect samples for every analysis. More specialized sample building will take place at Tier-1 and Tier-2 centres. Every Tier-1 centre receives a complete copy of the AOD — all of the streams. The streams merely control which events are close to which other events in which files.

2.2.5.4 TAG Production

AOD production jobs simultaneously write event-level metadata (event TAGs), along with “pointers” to POOL file-resident event data, for later import into relational databases. The purpose of such event collections is to support event selection (both within and across streams),

and later direct navigation to exactly the events that satisfy a predicate (e.g., a cut) on TAG attributes.

To avoid concurrency control issues during first-pass reconstruction, TAGs are written by AOD production jobs to files as POOL "explicit collections," rather than directly to a relational database. These file-resident collections from the many AOD production jobs involved in first-pass run reconstruction are subsequently concatenated and imported into relational tables that may be indexed to support query processing in less than linear time.

While each event is written to exactly one AOD stream, references to the event and corresponding event-level metadata may be written to more than one TAG collection. In this manner, physics working groups may, for example, build collections of references to events corresponding to (possibly overlapping) samples of interest, without writing event data multiple times. A master collection ("all events") will serve as a global TAG database. Collections corresponding to the AOD streams, but also collections that span stream boundaries, will be built during first-pass reconstruction.

Standard utilities provided by the database group make it possible to analyse the events in such collections by following the pointers to the corresponding events at sites that host the corresponding event data (e.g., all Tier-1 sites for AOD), or, alternatively, to run extraction jobs that iterate over all events in such a collection and extract (copy) the corresponding data into personal files that contain those events and no others, perhaps for shipment to smaller-scale facilities or personal computers. Such extraction is expected to be common at Tier-1 and Tier-2 sites.

Alternate models have been considered, and could also be viable. It is clear from the experience of the TeVatron experiments that a unique solution is not immediately evident. The above scenario reflects the best current understanding of a viable scheme, taking into account the extra constraints of the considerably larger ATLAS dataset. It relies heavily on the use of event collections and the TAG system. These methods are only undergoing their first serious tests at the time of writing. However, the system being devised is flexible, and can (within limits) sustain somewhat earlier event streaming and modestly overlapping streams without drastic technical or resource implications.

2.2.5.5 Reprocessing

The current model assumes that the new data will be reprocessed approximately 2-3 months after acquisition using the same software version but improved calibration and alignments. These will be obtained from continued study of the calibration stream data and also of the first-pass ESD. It is this 'offline' calibration process that sets the time scale for the reprocessing. A second reprocessing of the complete dataset, including the data from previous years, is envisaged at the end of data taking each year, using up-to-date algorithms and calibrations. The reprocessing will probably take place more frequently during the first couple years. In some cases it may be possible to reprocess starting from ESD rather than going back to the raw data.

It is assumed that the bulk reprocessing occurs at the Tier-1 facilities, and that the dominant access to RAW data will be through this scheduled and read-occasionally process. It is possible that the EF farm could be pressed into service for this activity; we believe that the architecture of the Tier-0 and EF farms should allow the repartitioning of resources between the two. However, such dual use is not assumed in the baseline Computing Model.

2.3 Data Analysis

The types of event data (SIM/RAW, ESD, AOD, TAG, DPD) are described in an earlier section. All the useful events acquired from the detector and those events from large production Monte Carlo samples will be reconstructed to produce ESD, AOD and TAG using the production process described in the previous section. In principle, a user wishing to perform data analysis can access any of this data but, in practice, the available resources (CPU, disk and bandwidth) for any one job will limit access to a small fraction. This is true not because most of the resources are dedicated to production but because there are many such analysts each typically submitting a series of jobs in an iterative analysis process.

It should be noted in the following that analysis in 'local' Tier-3 facilities (which may in practice be a fraction of a facility otherwise regarded as a Tier-2 or even a Tier-1), where the resource allocation and sharing is completely under local control for a local community, is not included. However, such Tier-3 facilities may form an appreciable additional resource in the overall ATLAS computing.

2.3.1 Analysis Procedures and Data Flow

The resources required by analysis jobs will vary widely and a combination of physics priority and fair share will be used to allocate resources and thus determine which jobs run and when. On the communal ATLAS Tier-2 resources, each user will be assigned a resource quota that can be extended with approval from a physics group. Jobs exceeding the quota assigned to an individual user can be submitted to the production system for processing in a more controlled manner with priority assigned by a physics group. All large-scale access to Tier-1 resources must be arranged through physics or detector groups, given the resource implications. A Grid computing system will enable processing to take place at remote sites and even multiple sites for a single job. The Grid model also makes the system extensible: non-ATLAS Grid resources can easily be utilized when available and needed.

The goal of the data organization is to enable users to identify the input dataset of interest and then to enable the processing system to gain efficient access to the associated data. Here "dataset" refers to some collection of data, possibly, but not necessarily, all the data in a single file or collection of files. These datasets are catalogued along with metadata specifying their content (ESD, AOD, …) and book-keeping data specifying their provenance and quality to enable users to make selections. The provenance also enables users to discover if the output dataset they intend to create already exists or at least appears in the catalogue.

The datasets appearing in the metadata catalogue are typically virtual, i.e. they have no specific location (e.g. list of files) holding their data. Instead there is a dataset replica catalogue, which provides the mapping to one or more concrete replicas for each virtual dataset. The processing system may choose from these the concrete dataset whose location provides the most efficient data access. The job description should also include the required content to ensure that the input dataset is suitable for processing. If the input dataset includes unneeded content, it may be replaced with a sub-dataset removing the need to stage unused data.

The POOL collection files make use of the ROOT infrastructure and so the AOD and ESD event headers and objects they reference are directly accessible once the files holding these objects are available. The distributed analysis system will typically stage all required files on a local disk before starting a processing job. If the data are sparse, i.e. the dataset does not reference all the

data in the files, the system may copy the referenced data (e.g. selected events) into new files to avoid transferring data that will not be accessed. A new concrete dataset may be formed from these files. This dataset is a concrete replica equivalent to the original. The new files and dataset may be catalogued for future processing.

Both ESD and AOD are stored in POOL event collection files and are processed using the ATLAS software framework, Athena. The TAG data are stored in relational tables as event attributes for these pool collections. The decreasing event size in the event model allow an analyst with a given set of resources, to process a much large number of AOD events than ESD or RAW events. In addition, the AOD is likely to be more accessible with a full copy at each Tier-1 site and large samples at Tier-2 sites. An analyst beginning with a sample containing a very large number of events can issue a query against the TAG data to select a subset of events for processing using AOD or ESD.

A typical analysis scenario might begin with the physicist issuing a query against a very large TAG dataset, e.g. the latest reconstruction of all data taken to date. For example, the query might be for events with three leptons and missing transverse energy above some threshold. The result of this query is used to define a dataset with the AOD information for these events. The analyst could then provide an Athena algorithm to make further event selection by refining the electron quality or missing transverse energy calculations. The new output dataset might be used to create an n-tuple for further analysis or the AOD data for the selected events could be copied into new files. A subset of particularly striking events identified in one of these samples could be used to construct a dataset that includes the ESD and perhaps even RAW data for these events. The physicist might then redo the electron reconstruction for these events and then use it to create a new AOD collection or n-tuple.

An actual analysis would be much more complicated with steps being repeated and perhaps requiring the addition of Monte Carlo signal and background samples. Large data samples (say 0.1 TB and larger) will be processed using the distributed analysis system where the user specifies an input data dataset and query or algorithm (also known as "transformation") to apply to this dataset and the processing system generates an output dataset. Each dataset may include event data and/or summary (histogram, n-tuple…) data. An event dataset may be represented in many ways: a deep copy of the included data, a copy of the relevant event headers, the tokens for these event headers, a list of event identifiers with reference to another dataset, or simply references to the transformation and input dataset (virtual data). The processing system decides which is most appropriate and where to place the associated data possibly with some guidance from the user. This enables the system to balance usage of the different resources (processing, storage and network).

2.3.2 The Analysis Model and Access to Resources

For the purposes of estimation of the required resources for analysis, the analysis activity is divided into two classes. The first is a scheduled activity run through the working groups, analysing the ESD and other samples, and thereby extracting new TAG selections, working-group enhanced AOD sets or n-tuple equivalents. The jobs involved would be developed at Tier-2 sites using small sub-samples in a 'chaotic' manner (i.e. not involving detailed scheduling). Running over the large data sets would be approved by working-group organizers. It is assumed there are ~20 physics groups at any given time, and that each will run over the full sample four times in each year. It is also assumed that only two of these runs will be retained, one current and one previous.

The second class of user analysis is chaotic in nature and run by individuals. It is mainly undertaken in the Tier-2 facilities, and includes direct analysis of AOD and small ESD sets and analysis of DPD. It is estimated that for the analysis of DPD, some 25 passes over 1% of the events collected each year would require only 92 SI2k per user. (It should be kept in mind that the majority of the user analysis work is to be done in Tier-3s.) Assuming the user reconstructs one thousandth of the physics events once per year, this requires a more substantial 1.8 kSI2k per user. It is assumed the user also undertakes CPU-intensive work requiring an additional 12.8 kSI2k per user, equivalent to the community of 600 people running private simulations each year equal to 20% of the data-taking rate. Such private simulation is in fact observed in some experiments, although it must be stressed that all samples to be used in published papers must become part of the official simulation sets and have known provenance. It is assumed that each user requires ~1 TB of storage by 2007/2008, with a similar amount archived.

From this perspective, the activities of the CERN Analysis Facility are seen to be those of a large Tier-2, but with a higher-than-usual number of ATLAS users (~100), without the simulation responsibilities required of a normal Tier-2. It is envisaged that there will be ~30 Tier-2 facilities of various sizes, with an active physics community of ~600 users accessing the non-CERN facilities.

2.3.3 Distributed Analysis System

The distributed analysis system will enable users to submit jobs from any location with processing to take place at remote sites. It will provide the means to return a description of the output dataset and enable the user to access quickly the associated summary data. Complete results will be available after the job finishes, while partial results will be available during processing. The system will ensure that all jobs, output datasets and associated provenance information (including transformations) are recorded in the catalogues. In addition, users will have the opportunity to assign metadata and annotations to datasets, as well as jobs and transformations to aid in future selection.

The distributed analysis system will typically split a user job request into a collection of subjobs, usually by splitting the input dataset. The results (output datasets) of the subjobs will be merged to form the overall output dataset. The distributed analysis system will decide how to split and merge datasets taking into account available resources and user requirements such as response time. Some fraction of resources will be dedicated to interactive analysis that enables users to examine (at least partial) results 10-100 seconds after job submission.

2.4 Simulation Process

Simulated data are assumed to be produced in the external Tier-2 facilities. Once produced, the simulated data must be available for the whole collaboration on an essentially 24/7 basis, as for real data. This requirement suggests that the simulated data should be stored at the Tier-1 facilities unless the lower tiers can guarantee the required level of access. However, it is assumed that all of the required derived datasets (ESD, AOD and TAG) are produced together at the same site, and then transported to their eventual storage location.

If the produced simulated data are to be shipped to the local storage site, this would contribute to the required connectivity to the regional Tier-1. If it is not shipped offsite, the bandwidth must also support the replication offsite on demand when the simulated data are analysed, or

else the local facility must also support (and be credited for) the processing of ATLAS analysis jobs from all regions. The relative merits of moving jobs to the data or data to the jobs are a key issue in the Grid, and the optimal strategy depends on the requested data volume, the CPU requirement of the job and the available network interconnect. In general, all the factors tend to suggest that the storage and analysis of simulated data are best handled through the Tier-1 facilities by default, although some larger Tier-2 facilities may wish to share this load, with appropriate credit.

In addition to the official datasets that are officially requested by ATLAS and available to all, there will be additional samples that will be generated locally to test and optimize the simulation procedures and support local analyses. These only enter in the Computing Model in terms of their resource requirements, and are accounted in the resource requirements per user in the Section 2.3.

2.5 Calibration and Alignment

Calibration and alignment processing refers to the processes that generate 'non-event' data that are needed for the reconstruction of ATLAS event data, including processing in the trigger/event filter system, prompt reconstruction and subsequent later reconstruction passes. These 'non-event' data (i.e. calibration or alignment files) are generally produced by processing some raw data from one or more sub-detectors, rather than full raw data, so e.g. Detector Control Systems (DCS) data are not included here. The input raw data can be in the event stream (either normal physics events or special calibration triggers) or can be processed directly in the sub-detector read-out systems. The output calibration and alignment data will be stored in the conditions database, and may be fed back to the online system for use in subsequent data taking, as well as being used for later reconstruction passes.

Calibration and alignment activities impact the Computing Model in several ways. Some calibration will be performed online, and require dedicated triggers, CPU and disk resources for the storage of intermediate data, which will be provided by the event filter farm or a separate dedicated online farm. Other calibration processing will be carried out using the recorded raw data before prompt reconstruction of that data can begin, introducing significant latency in the prompt reconstruction at Tier-0. Further processing will be performed using the output of prompt reconstruction, requiring access to AOD, ESD and in some cases even RAW data, and leading to improved calibration data that must be distributed for subsequent reconstruction passes and user data analysis.

2.5.1 Types of Processing

Various types of calibration and alignment processing can be distinguished:

- Processing directly in the sub-detector read-out system (the RODs). In this case, the processing is done using partial event fragments from one sub-detector only, and these raw data fragments do not need to be passed up through the standard ATLAS DAQ chain into the event stream (except for debugging). This mode of operation can be used in dedicated stand-alone calibration runs, or using special triggers during normal physics data-taking.

- Processing in the EF system, with algorithms either using dedicated calibration triggers (identified in the level 1 trigger or HLT), or 'spying' on physics events as part of the normal processing. In particular, an algorithm running at the end of a chain of event filter algorithms would have access to all the reconstructed information (e.g. tracks) produced during event filter processing, which may be an ideal point to perform some types of calibration or monitoring tasks. If the calibration events are identified at level 1 or 2, the event filter architecture allows such events to be sent to dedicated sub-farms, or even for remote processing at outside institutes.
- Processing after the event filter, but before prompt reconstruction. Event byte-stream RAW data files will be copied from the event filter to the Tier-0 input buffer disk as soon as they are ready, and could then be processed by dedicated calibration tasks running in advance of prompt reconstruction. This could be done using part of the Tier-0 resources, or event files could also be sent to remote institutes for processing, the calibration results being sent back for use in later prompt reconstruction, provided the latency and network reliability issues can be kept under control.
- Processing offline after prompt reconstruction. This would most likely run on outside Tier-1 or Tier-2 centres associated with the sub-detector calibration communities, leaving CERN computing resources free to concentrate on other tasks. RAW data, ESD and AOD will all be distributed outside CERN, though data from more than one centre would be needed to process a complete sample due to the 'round-robin' distribution of RAW and ESD to Tier-1 centres.

All of these processing types will be used by one or more of the ATLAS sub-detectors; the detailed calibration plans for each sub-detector are still evolving. The present emphasis is on understanding the sub-detector requirements, and ensuring they are compatible with the various constraints imposed by the different types of online and offline processing.

2.5.2 Calibration Streams

As discussed above, the output from the event filter will consist of four main streams: the principal physics stream, an express stream of 'discovery-type' physics, a calibration stream, and a diagnostic stream of pathological events. A first outline and incomplete proposal for the calibration stream is given below, though the details will evolve towards and beyond the start of data taking:

- An inner detector alignment stream, with 10-100 Hz of reconstructed track information (not raw data), processed in the event filter, and amounting to a maximum of 4 MB/second.
- A LAr electromagnetic calorimeter calibration stream, with 50 Hz of inclusive electron candidates identified in the event filter. All five time samples of the electromagnetic calorimeter would be written out, but only for the region around the electron candidate, amounting to 50 kB/event or 2.5 MB/s.
- A muon calibration stream, taking a muon chamber region of interest identified at level 1 and outputing muon (MDT and trigger chamber) hit data for auto-calibration at the full level 1 rate of O(10 kHz), corresponding to 6 MB/s. This may have a significant effect on the muon level 1 trigger for physics data taking, and may only be done for a few hours each week.

- Inclusive high p_T electrons and muons selected by the event filter at 20 Hz, with full event read-out (32 MB/s). All these events will also be written in the primary physics stream, but duplicating these data into separate calibration streams will greatly facilitate efficient access for global detector debugging, calibration and reconstruction tuning. This will be especially important during initial running, and it is anticipated that this stream would gradually be phased out as data-taking advances and experience is gained with handling the primary physics stream through event collections.

These streams sum to a total data rate of about 45 MB/second, dominated by the inclusive high p_T leptons, corresponding to 13% of the total bandwidth out of the event filter (200 Hz of 1.6 MB events). The RAW data for all these streams corresponds to 450 TB/year, not counting ESD and subsequent reprocessing passes (which will be frequent at least at the beginning). It is clear that only a fraction of this data will be kept on disk at Tier-0, and priority will have to be given to the most recent data. However, this should be acceptable as most of the data is only needed for short-term calibration and debugging activities that should be complete in a few days or weeks, at least once the initial start-up phase is completed.

2.5.3 Prompt Reconstruction Latency

Prompt reconstruction latency refers to the time between the data being taken and it being processed through all stages so as to be ready for user analysis. Assuming that event filter output nodes (SFOs) write 2 GB files, each one will fill and close such a file every five minutes, and this should be transferred to the Tier-0 input buffer disk within a further few minutes. The reconstruction and ESD production for each file will be done on a single processor, and is expected to take around five hours, with AOD production introducing a small extra latency.

The latency incurred in the preparation of conditions data is much more difficult to assess. Several sub-detectors have calibration constants that are expected to change significantly for each LHC fill, and these will be re-determined every fill or every day using dedicated calibration tasks running from the raw data or independently of the event stream (e.g. optical muon and inner detector optical alignment systems). It seems unlikely that the calibration processing to determine the first-pass calibration constants can be completed much sooner than 24 hours after the end of the fill, as it may require some preliminary reconstruction on dedicated calibration samples followed by verification on independent samples. A global optimization is needed amongst all sub-detectors to see what would be gained by a target of 12, 24 or 48 hours, balanced against the need for increased disk buffer storage at Tier-0 to avoid staging all the data to tape and then back in again for prompt reconstruction. The express physics stream would probably be processed more quickly, perhaps using the constants derived from the previous fill. Such a fast turnaround would also provide a sample of data useful for rapid data quality monitoring using fully reconstructed events. In general, considering possible failures in the calibration process and the potential problems introduced by delayed or off-site first-pass processing, a buffer corresponding to five days of input RAW data will be required for ATLAS.

2.5.4 Offline Calibration and Alignment

In principle, offline calibration and alignment processing is no different to any other type of physics analysis activity and could be treated as such. In practice, many calibration activities will need access to large event samples of ESD or even RAW data, and so will involve resource-intensive passes through large amounts of data on the Tier-1s or even the Tier-0 facility. Such ac-

tivities will have to be carefully planned and managed in a similar way to bulk physics group productions. At present, the offline calibration processing needs are not sufficiently well understood, though the recent definitions of the contents of DRD, ESD and AOD should help sub-detectors in defining what processing tasks are required, and how they will be accomplished.

2.6 Heavy Ion Data

It is foreseen that ATLAS will take data with Heavy Ion (initially Pb-Pb) collisions for approximately one month per year, starting in late 2008 [2-8]. Here we assume that the Heavy Ion data-taking period will last 10^6 seconds (50000 seconds/day for 20 days) each year. The trigger rate is effectively limited by the available bandwidth between the HLT and Tier-0 buffers (320 MB/s) and the event size (5 MB) to ~50 Hz. We take this as a reference number for the purpose of this document.

The Computing Model for Heavy Ion data is not the same as for p-p data. A significantly reduced Event Data representation is assumed, and the simulation and processing will be tailored to reflect this. The assumptions about data sizes and processing times are given in Table 2-3. Note that the simulated datasets will be a mixture of central, peripheral and 'model' events.

Table 2-3 The assumed heavy ions event data sizes for various formats, the corresponding processing times and related operational parameters.

Item	Unit	Value
Raw Data Size	MB	5
ESD Size	MB	3
AOD Size	kB	1000
TAG Size	kB	1
Simulated Data Size	MB	130
Simulated ESD Size	MB	6
Time for Reconstruction (1ev)	kSI2k-sec	200
Time for Simulation (1ev)	kSI2k-sec	20000
Time for Analysis (1ev)	kSI2k-sec	10
Event rate after Event Filter	Hz	50
Operation time	seconds/day	50000
Operation time	days/year	20
Operation time (2007)	days/year	0
Event statistics	events/day	$2.5 \cdot 10^6$
Event statistics	events/year	$5 \cdot 10^7$

There is no need foreseen for two reprocessings of the data in the year in which it is taken. instead, a small fraction will be processed during the data taking for control and monitoring, and then the full processing performed during the gap between p-p processing at the Tier-0. While it

makes a marginal impact on the computing model, it is assumed that only a subset of Tier-1s (perhaps only two) and a corresponding subset of Tier-2s will house and process the heavy ions data.

Another possibility is that Heavy Ion data are distributed only to the subset of Tier-1s hosting the communities of physicists most interested in those data, and processed only there. This model necessitates a study of the available network bandwidth to those computing centres and significantly increased resources to be made available in those centres, although of course the total amount of required resources would remain the same as in the "simplest" model above.

2.7 Commissioning the System

The data processing in the very early phase of data taking will be rather different to the steady state scenario described above. While the distribution and access to the data should be well-prepared and debugged by the various data challenges, there will still be a requirement for heightened access to raw data to produce the primary calibrations and to optimize the reconstruction algorithms in the light of the inevitable surprises thrown up by real data. The access to raw data is envisaged in two formats, RAW files and (if possible) DRD.

As will be seen in Chapter 7, the steady-state model has considerable capacity for analysis and detector/physics group files. There is also a significant planned capacity for analysis and optimization work in the CERN analysis facility. It is envisaged that in the early stages of data taking, much of this is taken up with a deep copy of the express and calibration stream data. For the initial weeks, the express data may be upwards of 20 Hz, but it is clear that averaged over the first year, it must be less than this. If this averages at 10 Hz over the full year, and we assume we require two processing versions to be retained at any time at the CERN analysis facility, this translates to 620 TB of disk.

It is also assumed that there will be several reprocessings of these special streams. The CPU involved must not be underestimated. For example, to process the sample 10 times in 6 months would require a CPU capacity of 1.1 MSI2k (approximately 1000 current processors). This is before any real analysis is considered. Given the resource requirements, even reprocessing this complete smaller sample will have to be scheduled and organized through the physics/computing management.

Groups must therefore assess carefully the required sample sizes for a given task. If these are small enough, they can be replicated to Tier-2 sites and processed in a more ad hoc manner there. Some level of ad hoc reprocessing will of course be possible on the CERN Analysis Facility.

The CERN Analysis Facility resources are determined in the Computing Model by a steady-state mixture of activities that includes AOD-based and ESD-based analysis and steady-state calibration and algorithmic development activities. This gives 1.1 PB of disk, 0.58 PB of tape and 1.7 MSI2k processing power for the initial year of data taking. This resource will initially be used far more for the sort of RAW-data based activity described in Section 2.3 and Section 2.5, but must make a planned transition to the steady state through the first year. If the RAW data activities continue on a large scale much longer, the work must be shared by other facilities. The Tier-1 facilities will also provide calibration and algorithmic development facilities throughout, but these will be limited by the high demands placed on the available CPU by reprocessing and ESD analysis.

There is large flexibility in the software chain in the format and storage mode of the output datasets. For example, in the unlikely event of navigation between ESD and RAW proving problematic when stored in separate files, they could be written to the same file. As this has major resource implications if it were adopted as a general practice, this would have to be done for a finite time only and on a subset of the data. Another option that may help the initial commissioning process is to produce DRD, which is essentially RAW data plus selected ESD objects. This data format could be used for the commissioning of some detectors where the overhead of repeatedly producing ESD from RAW is high and the cost of storage of copies of RAW+ESD would be prohibitive. In general, the aim is to retain flexibility for the early stage of data taking in both the software and processing chain and in the use of the resources available.

2.8 Summary

The baseline steady-state ATLAS Computing Model for the early years of data taking has been outlined. It presumes a well-planned, organized and maintained Virtual Distributed Computing Facility. Such a facility will support ATLAS production, simulation and analysis. A considerable degree of central organization will be required in the analysis activities given the volume of data to be processed in a world-wide system.

Consideration has also been given to the commissioning of the system with first data. It is essential that a considerable degree of flexibility both in data formats and in resource usage be retained to ensure rapid analysis of the early data and a swift understanding and calibration of the detector. Our continuing Data and Service Challenge activities are designed to refine the understanding of the Computing Model, and also to commission the required tools well in advance of the first data taking.

2.9 References

2-1 M. Aderholz *et al.*, *MONARC Phase 2 Report*, CERN-LCB-2000-001, March 2000.

2-2 S. Bethke *et al.*, *Report of the Steering Group of the LHC Computing Review*, CERN-LHCC-2001-004, March 2001

2-3 *The LHCC Computing Resources Review*, CERN-LHCC-2005-006, March 2005

2-4 ATLAS Collaboration, *ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report*, CERN-LHCC-2003-022, June 2003.

2-5 ATLAS Collaboration, *ATLAS Detector and Physics Performance Technical Design Report*, CERN-LHCC-1999-14 and CERN-LHCC-1999-15, June 1999.

2-6 D.G. Charlton *et al.*, *Report of the ATLAS Online–Tier-0 Task Force*, ATL-GEN-2004-002, December 2004

2-7 D. Rousseau *et al.*, *Report of the ATLAS AOD/ESD Definition Task Force*, ATL-SOFT-2004-006, December 2004.

2-8 ATLAS Collaboration, *Heavy Ion Physics with the ATLAS Detector*, CERN-LHCC-2004-009, March 2004.

3 Offline Software

3.1 Introduction and Requirements

The high-level goals of the ATLAS offline software are to process the events delivered by the ATLAS trigger and data acquisition system [3-1], to deliver the processed results to physicists within the ATLAS Collaboration, and to provide tools for them to analyse the processed information in order to produce physics results. Technical requirements include the processing time and memory consumption per event necessary to meet financial constraints, and the physics performance requirements encompasses the ability to reproduce faithfully details of the underlying physics processes based on the detector data.

The fulfilment of the processing requirements requires the provision of simulation tools whereby the performance of the software may be tested against a true knowledge of the underlying physics processes. This capability has also been used to aid in the design of the detector and data acquisition hardware.

The complexity and scale of ATLAS mean that the software must be highly modular and robust, and must furthermore be flexible enough to meet the needs of the experiment throughout its operational lifetime. There will no doubt be changes in the physics goals and even detector hardware during this period and the software must accommodate these.

The complexity of ATLAS and the underlying physics require a sophisticated real-time trigger system and components of the offline software should be capable of operating in the trigger environment alongside components designed specifically for it. In general it should be possible to migrate offline processing components into the high-level trigger environment as the physics goals of the experiment change over its lifetime.

The design and implementation of this software requires the establishment of a software development environment capable of supporting a distributed development model appropriate for the widely geographically dispersed ATLAS developer community, and the provision of testing and quality assurance tools to ensure that the software meets both the technical and physics performance requirements.

3.2 Methodology and Domain Decomposition

3.2.1 An Object-Oriented Methodology

ATLAS has adopted an object-oriented development methodology, based primarily on the C++ programming language, but with some components implemented using FORTRAN and Java. A component-based model has been adopted, whereby applications are built up from collections of plug-compatible components based on a variety of configuration files. This capability is supported by a common framework that provides common data-processing support. This approach results in great flexibility in meeting the basic processing needs of the experiment and in responding to changing requirements throughout its lifetime. The heavy use of abstract inter-

faces allows for different implementations to be provided, supporting different persistency technologies, or optimized for the offline or high-level trigger environments.

A rather light-weight development process has been adopted, with emphasis on guidance and use of automated tools and extensive testing and validation rather than a rigid enforcement of strict protocols. Experience has shown that such an approach is better matched to the primarily physicist developer community of a high-energy physics experiment than a heavy-weight process. However, there are clearly potential disadvantages to such an approach in the areas of robustness and reliability, and more emphasis has been placed on the formal development process in core areas of the software.

3.2.2 Domain Decomposition

Several orthogonal domain decompositions have been identified. The first spans the ATLAS detector subsystems:

- Inner detector, which itself is comprised of the pixel detector, the silicon strip detector and the transition radiation tracker.
- Liquid argon calorimeter.
- Tile calorimeter.
- Muon spectrometer.

The primary data-processing activities that must be supported for all of these detector subsystems are:

- Event generation.
- Simulation.
- Pile-up.
- Digitization.
- Detector reconstruction.
- Combined reconstruction.
- Physics analysis.
- High-level triggering.
- Online monitoring.
- Calibration and alignment processing

Further domain decompositions cover the infrastructure needed to support the software development activity, and components that derive from the overall architectural vision. The overall structure is the following:

- *Framework and Core Services*. A common event processing framework has been adopted based on plug-compatible components and abstract interfaces. A variety of services such as scripting, detector geometry, and graphics have been developed in support of this framework. The overall architecture and common framework are described in Section 3.3, "The Athena Framework". Particular core services are described in detail in Section 3.4,

"Data Access Model", Section 3.5, "Detector Description" and Section 3.6, "Graphics and Event Display"

- *Event generators, simulation, pile-up and digitization*. These are described in Section 3.8, "Simulation"
- *Event selection, reconstruction and physics analysis tools*. These are described in Section 3.9, "Event Selection and Reconstruction" and Section 3.10, "Physics Analysis Tools"
- *Calibration and alignment*. The high-level requirements for sub-detector calibration and alignment are described in Section 3.7, "Calibration and Alignment"
- *Infrastructure*. This provides the services that support the software development process including the source code repository, tools for building coherent releases of the software, and for testing and validating those releases, and documentation tools and services. This is described in detail in Section 3.14, "Software Infrastructure"

3.3 The Athena Framework

3.3.1 Introduction

The Athena framework [3-2] is an enhanced version of the Gaudi framework [3-3] that was originally developed by the LHCb experiment [3-4], but is now a common ATLAS–LHCb project and is in use by several other experiments including GLAST and HARP. Athena and Gaudi are concrete realizations of a component-based architecture (also called Gaudi) which was designed for a wide range of physics data-processing applications. The fact that it is component-based has allowed flexibility in developing both a range of shared components and, where appropriate, components that are specific to the particular experiment and better meet its particular requirements.

3.3.1.1 Design Principles

Major design principles that influenced the development of Athena were:

- Abstract interfaces. These allow for the provision of different implementation providing similar functionality but optimized for particular environments (e.g. high-level trigger and offline, different persistency technologies). They also allow for easy manipulation of groups of components sharing a common interface.
- Extensive use of dynamic libraries and loading implement the Gaudi component architecture.
- A clear separation between data and algorithms. For example, the interface that allows access to the data of a track (e.g. its 4-momentum) is separated in different classes (and usually different packages) from the algorithmic code responsible for finding and building tracks given a list of track segments. At first sight, this separation appears to violate one of the main principles of object-oriented software, encapsulation, but the opposite is actually true. By separating the complex algorithmic code that is responsible for the construction of the track objects from the much simpler and more stable client view of a track, the dependencies between the producer of a track object and its consumers are drastically reduced. This approach facilitates, for example, the transparent change at run time of the

track-finding strategy without having to recompile or to reconfigure downstream particle identification algorithms that are clients of track objects.

- The recognition that there are many types of data having different lifetimes within the software. Event data is directly associated with an ATLAS event, but detector information such as the geometry, alignment and electronics calibrations is in general stable across many physics events and therefore has a different lifetime. Similarly, statistical data accumulated in histograms is long-lived compared to the data of an individual event.
- A clear separation between persistent and transient data. In general the algorithmic code operating on the data should be independent of the technology used to store it, which might vary over the lifetime of the experiment, or depend upon the local environment (e.g. within the trigger data acquisition and offline environments).
- Recognition of multiple types of developer. Thus the code written by the end-user physicist should be exposed to relatively few, simply interfaces, and be isolated from that written by the tracking specialist or persistency expert.
- Attempt to re-use components and interface standards where feasible.

3.3.1.2 Major Component Abstractions

The major components that have been identified within the architecture are shown in Figure 3-1. This shows component instances and their relationships in terms of navigability and usage.

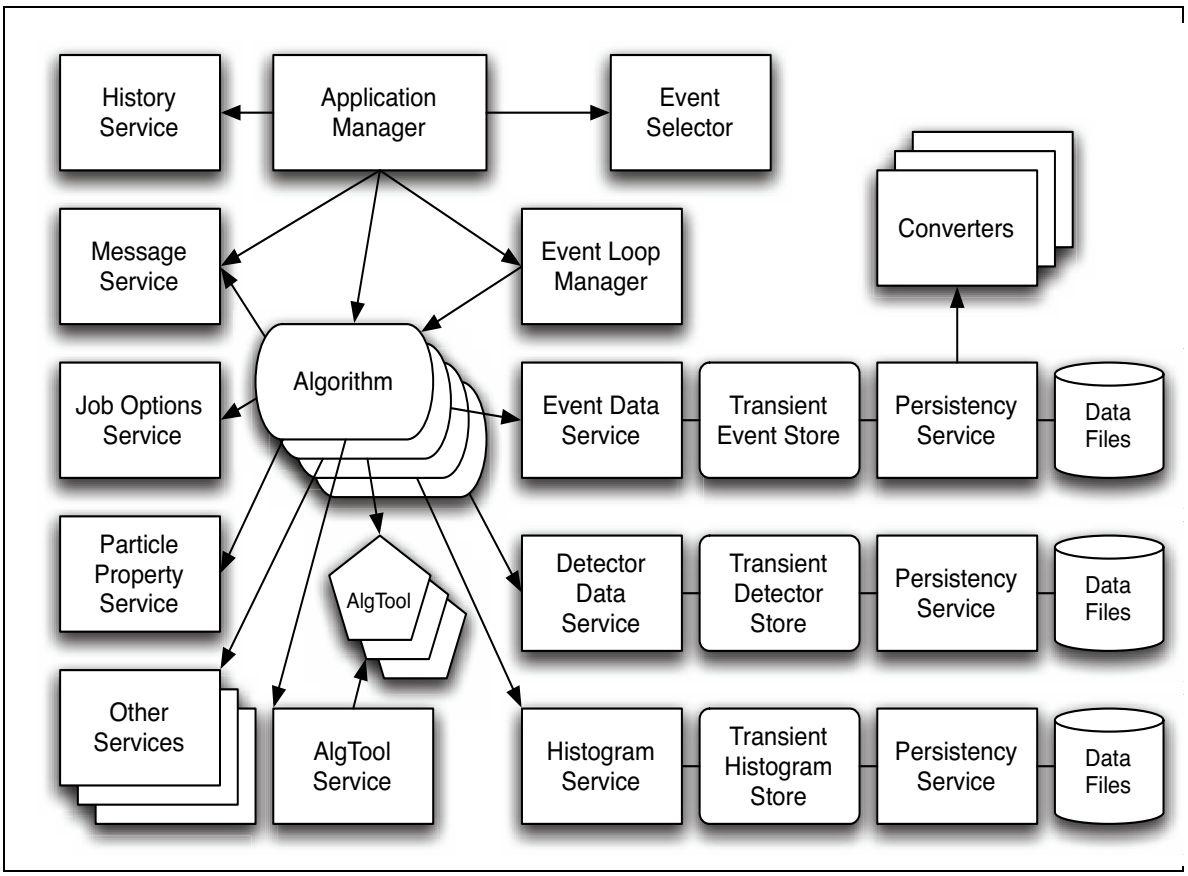

Figure 3-1 Athena Component Model

Major components are:

- **Application Manager**. The application manager is the overall driving intelligence that manages and coordinates the activity of all other components within the application. There is one instance of the application manager and it is common to all applications.
- **Algorithms and Sequencers**. Algorithms share a common interface and provide the basic per-event processing capability of the framework. Each Algorithm performs a well-defined but configurable operation on some input data, in many cases producing some output data.

  A Sequencer is a sequence of Algorithms, each of which might itself be another Sequencer, allowing for a tree structure of processing elements. A *filter* Algorithm can indicate that the event being processed fails to meet its filter criteria and inhibit the processing of downstream Algorithms.
- **Tools**. A Tool is similar to an Algorithm in that it operates on input data and can generate output data, but differs in that it can be executed multiple times per event. In contrast to Algorithms, Tools do not normally share a common interface so are more specialized in their manipulation. Each instance of a Tool is owned, either by an Algorithm, a Service, or by default by the AlgToolSvc.
- **Transient Data Stores**. The data objects accessed by Algorithms are organized in various transient data stores depending on their characteristics and lifetimes. The event data itself is managed by one store instance, detector conditions data, such as the geometry and alignment, by another store, etc.
- **Services**. A Service provides services needed by the Algorithms. In general these are high-level, designed to support the needs of the physicist. Examples are the message-reporting system, different persistency services, random-number generators, etc.
- **Selectors**. These components perform selection. For example, the Event Selector provides functionality to the Application Manager for selecting the input events that the application will process. Other types of selectors permit the selection of objects within the transient data stores.
- **Converters**. These are responsible for converting data from one representation to another. One example is the transformation of an object from its transient form to its persistent form and vice versa.
- **Properties**. All components of the architecture can have adjustable properties that modify the operation of the component. Typically these are basic types (integer, float, etc.), but can also be specified as having upper and lower bounds.
- **Utilities**. These are C++ classes that provide general support for other components.

3.3.1.3 Concrete Athena Services

One of the strengths of a common architecture is the accumulation of services useful to developers of algorithmic code. To start with, the Athena developer has available an array of core “system” services for job configuration, message logging, error handling, job tracing, performance monitoring and resource management.

- **Job Option Service.** The JobOptionSvc is a catalogue of user-modifiable properties of Algorithms, Tools and Services. As an example, the value of a cone radius cut "ConeR" in

the JetConeFinderTool named “ConeJets.ConeFinder” can be set either from a job-option file or from the Athena interactive prompt (see Section 3.3.1.4, "Scripting") by:

ConeJets.ConeFinder.ConeR = 0.7

Default values are set in the Algorithm, Tool or Service itself.

- **Logging and Error-handling**. The MessageSvc controls the output of messages sent by the developers using a MsgStream. The developer specifies the source of the message (its name) and the message verbosity level. The MessageSvc can be configured (using the JobOptionSvc) to filter out messages coming from certain sources or having a high verbosity level. Future implementations of MessageSvc are expected to log messages to a production or online database and to send critical error messages also to the (production or DAQ) operator console.

  When an Algorithm throws an Exception during the execute phase, it is caught and passed on to the Exception Service, which allows the users to manipulate the outcome. The service can be configured via job options to change the meaning of an exception from a failure to a success or recoverable error, to the status code of an associated GaudiException, or to re-throw the exception to a lower level, on a per-Algorithm or global basis.

- **Performance and Resource Monitoring**. The AuditorSvc and the ChronoStatSvc manage and report the results of a number of Auditor objects, providing statistics on the CPU and memory usage (including potential memory leaks) of Algorithms and Services. A TimeKeeper service monitors the remaining CPU time available to a job running in batch and terminates the job when it estimates that there is not enough time left to process a new event.

In addition to these generic tools, Athena provides many domain-specific tools, to record data-processing history, manage event and detector data, manage random-number generation, histogramming and n-tuples, and provide access to PDG [3-5] particle properties.

- **HistorySvc**. The ability to recreate the initial conditions of a particular event-processing job is of utmost importance. The History Service records the state of all Services, Algorithms, AlgTools, and run-time parameters and environment variables at the beginning of the job, using the appropriate type of History Object. Furthermore, whenever an object is registered with StoreGate[1], it creates an associated History Object that records the object type, classID and store that the object is saved into, as well as using the AlgContext Service to determine which Algorithm was being executed when the object was created. All these objects are linked in a hierarchical manner, allowing the user to navigate from the DataHistory Object all the way back to the JobHistory Object.

- **Histogramming and N-tuples**. The Histogramming and n-tuple services allow one to book, fill, manipulate and analyse histograms and n-tuples from within the Gaudi framework. Pre-existing histograms and n-tuples can also be imported from persistent storage. These services use abstract interfaces, so that the concrete implementation of each service can be changed if necessary, as has already been done with the transition from HTL [3-6] to ROOT [3-7] for the implementation of the histograms.

- **Access Time-Varying Data**. The Interval of Validity Service (IOVSvc) manages the automatic updating of conditions data. Users can request a smart pointer to the appropriate conditions data, knowing that it will always be kept up to date without their intervention.

1. StoreGate is described in Section 3.4.2.

Furthermore, callback functions can be registered with the service, so that when a conditions object enters a new range of validity, the callback function is triggered, allowing users to recalculate cached information. Callback functions are registered in a hierarchical manner, and are triggered according to their dependency tree. Users can choose when to check the ranges of validity – at every event, at the start of every run, or just once at the beginning of the job – to customize the level of database access, an important consideration for trigger-level activities. Also, users can force the service to preload the values of all ranges and related conditions objects at the beginning of the job, to eliminate all further database access during the course of the job.

- **Random Numbers**. The ATLAS Random Number service uses the RanecuEngine of CLHEP [3-8], and supports multiple streams, each stream being able to provide an independent sequence of random numbers. The initial seeds of each stream can be set via job options.

3.3.1.4 Scripting

There are two primary motivations for scripting within the Athena framework. The first is for configuration, taking advantage of the component architecture to build applications that are tailored for particular capabilities from a pool of available components. Thus the configuration should be able to specify the set of Algorithms and Services that are to be used, modify their behaviour by specifying overrides to the default Property settings, and specify any needed input or output files.

The second motivation is to provide support for interactivity, allowing the user to modify a static configuration and allow for rapid feedback from changes to, for example, tracking road widths, p_T thresholds and rapidly reprocessing a set of events so as to optimize the configuration for a later job over a larger event sample.

Athena uses a common scripting language, Python [3-9], for both the configuration and interactivity roles. So-called “Python bindings” are provided to its core abstractions, in particular the application manager, services, algorithms, histogramming, data store, etc. Scripting provides support not only for configuration and interactive use, but also for steering/control and interoperability with alternative front-ends to Athena. (The latter are not described further here.)

Python is a popular, open-source, dynamic language with an interactive interpreter. Like C++, Python is a multi-paradigm language and provides such concepts as modules, classes, exceptions and very high-level dynamic data types, etc. There are interfaces to most system calls, a large set of standard libraries, and bindings to many existing third-party applications and libraries. Initial scripting support for Athena was provided by an implementation of a Python scripting service conforming to the IRunable Service component abstraction. However, initial experience indicated that greater flexibility could be had by complementing this with a Python script that starts and runs the Athena Application Manager directly, using Python bindings to the core Athena abstractions.

Athena jobs are configured by specifying the set of Algorithms, Services, and other components to be used, as well as their Properties. This is done through the scripting interface, which transports the settings to the appropriate Athena components, or provides place-holders until the components are available. Since Python is a complete programming environment, end-users and package providers can write higher level interfaces on top of the Athena configuration facilities. This is often done to provide (semi-)automatic configuration based on a couple of basic, or initial settings.

The main Athena Python control script is simply a user interface on top of a set of scripts and modules that provide a programming interface at a more abstract level of steering than do the Athena component interfaces. It parses user command-line options, loops over configuration files, and sets some common defaults. The scripts/modules, all written in Python, with the code that performs the actual work, such as reading the configuration files, responding to settings, etc., are used in conjunction with other main Athena programs as well (in particular athena.C, and athenaMT).

The scripting facilities support interactive use of Athena; once the Python prompt has been reached, typically upon completion of the initial application configuration, actual instances of the components can be accessed, thus allowing their properties to be examined and modified. New components, possibly written in Python, can be loaded and configured, objects in the transient data store can be examined and their data can be plotted, and the event processing loop can be run completely in Python. New functionality can be built on top of the interactive layer, using third-party packages or parts of the Python standard libraries (e.g. the Python SimpleXMLRPCServer). An interactive physics analysis environment based on Python is under development.

3.4 Data Access Model

3.4.1 Data Objects and Algorithms

Using computing science terminology, the Gaudi software architecture belongs to the blackboard family: data objects produced by knowledge modules (called Algorithms in Gaudi) are posted to a common in-memory database (or "blackboard") from where other modules can access them and produce new data objects.

This model greatly reduces the coupling between the Algorithms containing the algorithmic code for analysis and reconstruction, in that one Algorithm does not need to know which specific Algorithm can produce the information it needs nor which protocol it must use to obtain it (the "interface explosion" problem described in component software systems). Algorithmic code is known to be the least stable component of software systems and the blackboard approach has been very effective at reducing the impact of this instability, from the Zebra system of the FORTRAN days to the InfoBus architecture for Java components. The trade-off of this separation between data and algorithm objects is the need for algorithm objects to identify data objects to be posted on or retrieved from the blackboard. It is crucial to develop a data model optimized for the required access patterns and yet flexible enough to accommodate unexpected ones.

3.4.2 StoreGate: the ATLAS Transient Data Store

The Transient Data Store (TDS) is the blackboard of the Gaudi architecture: an Algorithm creates a data object and posts it on to the TDS to allow other Algorithms to access it.

Once an object is posted on to the store, the TDS takes ownership of it and manages its lifetime according to preset policies, removing, for example, a TrackCollection when a new event is

read. The TDS also manages the conversion of a data object from/to its persistent form and provides therefore an API to access data stored on persistent media.

StoreGate is the ATLAS implementation of the TDS. It manages the data objects in transient form, it steers their transient/persistent conversion and it provides a dictionary allowing to identify and retrieve data objects in memory. The StoreGate design and implementation was largely driven by a few design concepts.

- *Work with User Types*. The success of the STL [3-10] and of other public domain template libraries means that it has become vital to design an open system that can work with generic types that export an interface, in particular the STL containers, rather than forcing data objects to import a common interface. StoreGate adapts its behaviour to the functionality each data object exports. The only StoreGate-imposed constraint on a data object is to be an STL *Assignable* type.

- *Avoid User-Defined Keys*. The disadvantage of the data/algorithmic object separation is the need for algorithmic objects to identify data objects to be posted on or retrieved from the blackboard. It is crucial to develop a data model optimized for the required access patterns and yet flexible enough to accommodate the unexpected ones.

  StoreGate addresses this problem with a two-step approach: it defines a natural identifier mechanism for data objects and it transparently associates to each data object a default value of this identifier allowing developers to register and retrieve data objects without having to identify them explicitly.

  The first component of the identifier is the data object type. Experience shows that HEP developers tend to group the objects they work on into collections, most often STL vectors. As a result the TDS will often contain a single instance of a data object type (e.g. a `TrackCollection)` or several related ones (e.g. a `TrackCollection` for each component of the Inner Detector). The StoreGate retrieve interface covers these two use cases.

  Type-based identification is not always sufficient. For example the TDS may contain several equivalent instances of a TrackCollection produced by alternative tracking algorithms. Therefore a second component has been added to the identification mechanism: the identifier of the Algorithm instance that produced the desired data object. In the spirit of working with user types, StoreGate allows developers to augment this history identifier with a generic key optimized for their access patterns.

- *Control Object Access and Creation*. The TDS is the main channel of communication among modules. A data object is often the result of a collaboration among several modules.

  StoreGate allows a module to use transparently a data object created by an upstream module or read from disk. A *Virtual Proxy* defines and hides the cache-fault mechanism: upon request[1], a missing data object instance can be transparently created and added to the TDS, presumably retrieving it from a persistent database or, in principle, even reconstructing it on demand.

  To ensure reproducibility of data processing, a data object should not be modified after it has been published to the store. The same handle/proxy scheme is used to enforce an ``almost const'' access policy: modules downstream of the publisher are only allowed to retrieve a constant iterator to the published object.

1. Currently the proxy uses lazy instantiation (i.e. the object is created only when the handle is dereferenced).

- *Support Inter-Object Relationships*. StoreGate supports unidirectional inter-objects relationships, or links. A link is a persistifiable pointer. If the linked object is a data object then the handle/proxy mechanism described above is also used to implement the link. But typically links will refer to objects that are not data objects but are contained within a data object. StoreGate knows how to get to the container and the container knows how to return an element given its index. The job of the link is to find out the value of the index, persistify it and, later on, pass it on to the container and get back the linked object.

3.4.3 Data Objects

As mentioned earlier, StoreGate is designed to work with user types rather than requiring them to implement a C++ interface. Basically any STL *Assignable* can be stored into StoreGate and hence is a Data Object.

A Data Object is a struct or class that encapsulates and "publishes" the result of some arbitrarily complex processing performed by one or more Algorithms. A Data Object should present a predictable, stable and efficient interface to client Algorithms.

StoreGate uses a compact, technology-independent mechanism to describe object types with two integer identifiers: a `CLID` and a `VERSION`. `CLID` is a 16-bit integer which uniquely identifies an object type across all ATLAS software for the lifetime of the experiment. If an object changes in a non-backward-compatible way a new `VERSION` number must be assigned to it[1].

Data Objects recorded to StoreGate are subsequently owned by StoreGate and the creator must not delete them.

3.4.3.1 Using Containers as Data Objects

Two different types of containers are supported by StoreGate:

- A *Value Container* is a container that owns its elements "by-value": the elements cease to exist when the container is deleted. For example the liquid argon calorimeter cell reconstruction may add the cells it makes to a `LArCellContainer` that is later recorded to StoreGate. When a `LArCellContainer` is deleted (typically at the end of event processing) all `LArCell` objects it contains are also deleted.
- A *View Container* is a container of object references. The referred-to objects are not owned by the View Container and will, in general, continue to exist after the View itself is deleted. As an example, the list of cells which were used to reconstruct a photon is a View on the container(s) of reconstructed calorimeter cells. A View Container that does not need to be persistified can be implemented using plain C++ pointers. A persistifiable view should be implemented using `DataLinks`[2].

3.4.4 Accessing Data Objects

StoreGate provides the following support for access to Data Objects managed by it.

1. The `VERSION` number is not yet used by StoreGate.
2. DataLinks are described in Section 3.4.5

- *Recording a Data Object*. In order to record a Data Object, StoreGate must be provided with a pointer to the object instance (from which the type is derived), and with a key. The combination of type and key must be unique. Once a Data Object has been recorded, StoreGate takes ownership of it.
- *Locking a Data Object.* Once a Data Object has been recorded it is locked by default such that it cannot be modified by downstream Algorithms. This can be overridden if necessary.
- *Retrieving a Data Object*. Data Objects in StoreGate are retrieved by type and key. StoreGate sets a pointer to the requested object(s). StoreGate supports the concept of automatic retrieval of Data Objects from a persistency service: if the requested Data Object has not yet been read from disk, StoreGate will message the persistency service and the appropriate converter[1] will create the Data Object.

  StoreGate supports the retrieval of a single keyed instance of an object as well as providing iterators to allow retrieval of all objects of a given type.
- *Checking if a Data Object is in the store*. StoreGate provides methods to check whether a given Data Object has already been stored.

3.4.5 Using DataLinks to Persistify References

In C++ associations among objects are described using pointers or references. For example, a Cluster object may refer to its list of associated Cells by holding a vector of Cell pointers. Unfortunately a plain C++ pointer can not be simply written out and read back from disk; it is valid only within the context of a running job since it refers to a virtual memory location.

To address this limitation `DataLink` and `ElementLink` have been introduced, being two class templates which can be dereferenced like a pointer and can be read and written using various persistency mechanisms. The `DataLink` template allows one to reference a single Data Object, using its unique type/key combination. `ElementLink` is used to reference an element of a container recorded in StoreGate.

3.5 Detector Description

The description of the ATLAS detector relies on two main software components: first, a relational database which implements a schema capable of hierarchical version control, and second, a set of geometrical primitives in terms of which the ATLAS detector is described.

A detector version tag, described in Section 3.5.1, is specified at simulation time, or at the time that real data is recorded, and that version is persistified along with the event data. The tag is used as a key into the relational database in order to retrieve a configuration, including all switches governing the detector geometry, and other properties related to the detector itself; in practice there is a fine line between certain types of detector description data and calibration data: e.g. magnetic fields. All data related to detector description should (ultimately) reside in that database.

1. Converters are described in Section 3.3.1.2.

The detector description is then built from the primary numbers describing the detector and switches. It includes all material geometry (the part which is seen directly by the simulation engine, Geant4 [3-11]) in addition to the readout geometry; for example, implant layers within silicon detectors; projective towers in the calorimeters, and similar. The geometry kernel provides mechanisms for expressing the geometry description into a minimal amount of memory. Subsystem developers have been made aware of memory optimization techniques. The current requirement for the entire ATLAS detector is 48 MB.

The detector description supports misalignment, both at simulation time or at reconstruction/analysis time. These misalignments are obtained from an alignment database and injected into the geometry description.

The IOVSvc is intended to be used during data-taking operations to map the time and date onto particular detector geometries. This does not interfere with the geometry tag definitions or the management of such tags, but can be considered as a separate, higher-level activity.

It should be noted that much of the work of defining the detector geometry is carried out on a subsystem-by-subsystem basis, where considerable customization is possible. Only the common aspects of detector description are described here. The following sections describe the hierarchically versioned relational database schema (HVS), and the geometry kernel which provides the set of geometrical primitives used to store the detector configuration.

3.5.1 ATLAS Geometry DB Schema - the Hierarchical Versioning System (HVS)

All primary numbers in the Geometry DB are logically grouped into ATLAS subsystem-specific “structures”, which map to rows within data tables. The structures are logically grouped into “directories”. Each directory can simultaneously contain child structures and subdirectories, and thus form a kind of hierarchy. In the Hierarchical Versioning System (HVS) the directories are called branch nodes and the structures leaf nodes. Characteristics of the tagging schema are:

- Both the branch and leaf nodes can be tagged.
- The branch node tag is comprised of the tags of its child nodes (branch and leaf).
- The leaf node tag is comprised of a set of corresponding data table records (structure instances).
- Each data table record can be tagged more than once. Also each tag of HVS node can be included into more than one tag of its parent node.
- It is possible to lock the tags. The contents of a locked tag cannot be changed any more. That means, in particular, that the data table records included in a locked tag cannot be updated or deleted.

The relational database schema of the Geometry DB can be decomposed into two main components:

- **Hierarchical Versioning System (HVS)** component, which implements the logical organization of primary numbers into HVS nodes. The HVS component includes descriptions of all HVS nodes, their tags and also mother-child relationships between tags;
- **Data** component, which actually holds the primary numbers for ATLAS Detector Description.

The HVS component consists of three tables:

- **HVS_NODE**. This table lists all HVS nodes in the database. The information about mother–child relationship between nodes is also kept within this table;
- **HVS_TAG2NODE**. This table lists all existing tags for all HVS nodes;
- **HVS_LTAG2LTAG**. This table lists mother–child relationships between HVS node tags.

Each HVS leaf node (structure) has two tables on the data component side:

- **Data table**. The columns of this table correspond to the structure fields by means of names and value types, plus one additional column to identify uniquely table records;
- **Relation table.** This table makes a connection between structure data table and HVS component by referencing the HVS_TAG2NODE table.

To manipulate the data in the Geometry DB effectively we have introduced three user accounts: Administrator, Writer and Reader. The person responsible for an ATLAS subsystem inserts the data into the Data component tables directly, executing SQL scripts via the SQLPLUS command line utility. For this purpose the Writer account is used.

Various manipulations with HVS nodes and tags can be done through a password-protected interactive Web interface. These operations include: creating new HVS node (branch and leaf); creating new HVS tag, deleting unlocked records from data tables, collecting tags and tag locking.

To read the data from the Geometry DB within Athena applications a dedicated Athena service, RDBAccessSvc, has been developed, which resides in the Database/AthenaPOOL container package. The RDBAccessSvc has been developed using the POOL Relational Access Layer, which provides the mechanism of uniform access to the data residing in the different RDBMS (Oracle, MySQL, SQLite). The RDBAccessSvc service allows simple manipulations with database connection (connect/disconnect) and also provides a uniform access to the versioned data in the Data component tables via the recordset objects. The recordset represents a snapshot of some data table, which includes only the records corresponding to the requested tag.

3.5.2 The Geometry Kernel (GeoModel)

The GeoModel toolkit is a library of geometrical primitives that can be used to describe detector geometries. The toolkit is designed as a data layer, and especially optimized in order to be able to describe large and complex detector systems with minimum memory consumption. Some of the techniques used to minimize the memory consumption are: shared instancing with reference counting, compressed representations of Euclidean transformations, special nodes which encode the naming of volumes without storing name-strings and, especially, parametrization through embedded symbolic expressions of transformation fields. A faithful representation of a GeoModel description can be transferred to Geant4. Native capabilities for geometry-clash detection and for material integration are foreseen for the near future. GeoModel's only external dependency is CLHEP.

The GeoModel toolkit provides application developers with a complete set of mechanisms for the description of the entire ATLAS detector with minimal memory consumption. The main purpose of GeoModel is to support a central store for detector-description information that can be accessed by two main clients – simulation and reconstruction programs. GeoModel provides a scheme for accessing both the raw geometry of a detector and arbitrary subsystem-specific ge-

ometrical services synchronized to the raw geometry, while incorporating time-dependent alignments. An automatic version detection system has also been developed.

In ATLAS, visual debugging tools developed with the use of the Open Inventor toolkit complement the GeoModel toolkit. These debugging tools include an interactive Geometry Browser[1] that allows various manipulations with volumes: navigation of volume hierarchy, iconization, printing of volume characteristics such as mass, shape dimensions, name and copy number. When used together with simulated data the Browser can also visualize tracks and hits on top of raw geometry.

In the following sections the design principles of the GeoModel toolkit are described, together with memory optimization techniques and also the mechanism for converting GeoModel descriptions to Geant4.

3.5.2.1 Material Geometry

Material geometry consists of a set of classes that resembles closely the Geant4 material geometry classes. These classes, however, are designed to take a minimal size in memory. This requirement determines the basic data structure used to hold the data for the geometry description. That structure is a graph of nodes consisting of both physical volumes and their properties (name, identifier, transformation in local coordinate system). The tree is built directly and accessed in a way that provides users with access to volumes and, simultaneously, to the properties accumulated during graph traversal that apply to the volumes.

Physical volumes are the main building blocks of the geometry graph. The structure of a physical volume consists of an associated logical volume (describing just shape and material) and a set of child physical volumes and their properties. In GeoModel two types of physical volumes are identified:

- Regular Physical Volumes, designed to be small;
- Full Physical Volumes, designed to hold in cache complete information about how the volume is located with respect to the world volume, its formatted name string and other important information.

When one adds a transformation in the graph, it controls the position of the subsequent volume with respect to the parent. If one adds more than one transformation to the volume before adding a parent, they will be applied successively. The last transformation to be added is applied first to the child. Like physical volumes, transformations come in two types:

- Regular transformations, designed to be small;
- Alignable transformations, which allow the addition of misalignments to the system. Misaligning a transformation changes the position of all volumes “under” the transformation and clears the absolute location caches of all full physical volumes.

GeoModel includes also three mechanisms for giving names to physical volumes:

- Do nothing; the volume will be called “ANON”;
- Add a Name Tag object to the graph before adding a volume; the next volume to be added will be given the Tag’s name;

1. The Geometry Browser is described in Section 3.6.3, "HEPVis/v-atlas".

- Add a Serial Denominator object to the graph before adding more volumes; the volumes will be named according to the base name of the Denominator, and given a serial number 0, 1, 2 …

3.5.2.2 Readout Geometry

The readout geometry layer consists of geometrical information that is not declared directly to the particle-tracking engines (Geant4), for example: projective towers within a calorimeter, or the boundaries of ion implant layers in silicon detectors. Information such as the position of the boundaries of these regions is not required in the simulation of basic physics processes, though it certainly is required in the digitization, and possibly hit-making phase of simulation (Sensitive Detectors).

Detector-specific geometrical services can and should include services that derive from the basic raw and readout geometry of the detector. Such services could include point-of-closest-approach calculations, global-to-local coordinate transformations, calculations that compute the total number of radiation lengths within a cell etc.

The description of the detector-readout system can be realized with the use of Detector Elements. The detector element has a required association with a piece of material geometry (full physical volume), and has access to that piece. The rest of the interface – all of the geometrical services discussed above, such as the boundaries of implant layers, strip pitches, etc., can be placed in the detector element.

3.5.2.3 Interface to Detector Description Clients

In GeoModel applications, the Detector Manager objects play a central role in providing an interface to Detector Description clients. Detector Managers manage raw and readout geometry and are responsible for providing a fast mechanism for accessing the detector elements in a detector-specific way.

Thus in general the subsystems developers have a lot of flexibility, but need to devise an interface to both the detector manager and the detector element that satisfies their needs. The basic framework requires only that

- Special Detector Factory objects create a physical volume tree;
- They associate readout elements to certain physical volumes.

Additional readout information appears in the interface to the detector manager and the detector element.

3.5.2.4 Memory Optimization Techniques

The requirement of minimizing the memory consumption has resulted in a design in which objects in the detector description can be re-used. This is called shared instancing. It essentially means that an element, compound volume, or entire tree of volumes may be referenced by more than one object in the detector description. The following use-cases for shared instancing exist: a small number of physical volumes can share the same logical volume, the same name tags and identifier tags can label different volumes, and the transformations can be used more than once in the geometry graph.

GeoModel includes some other memory optimization techniques: Serial Denominators can generate name strings for physical volumes so that the memory does not fill up with nearly identical ASCII name tags; "tiny" transforms (where the footprint for a simple translation along z, for example, is the size of one floating point number) reduce the size requirement for most transformations; finally, volumes can be parametrized (this mechanism is discussed in the next section).

3.5.2.5 Parametrizations

Parametrizations are mathematical recipes for creating volumes. There are three main ingredients to these recipes:

- GENFUNCTIONS, which are mathematical function-objects; they allow function arithmetic to be performed in the same way as floating-point arithmetic.
- TRANSFUNCTIONS, which, together with GENFUNCTIONS and HepTransform3D, allow elements of the Euclidean group (i.e., rigid body transformations) to be expanded and parametrized.
- Serial Transformers, a GeoModel graph node, which allow a particular physical volume to be placed according to a TRANSFUNCTION expansion of a rigid-body transformation.

This combination allows a single entity to be placed on the geometry graph which will then appear to contain multiple physical volumes at different locations. However, only the memory of a single physical volume and a TRANSFUNCTION is actually allocated.

3.5.2.6 Interface to Geant4, the Geo2G4 Translator

In order to run the Geant4 simulation of a detector described in GeoModel it is necessary to build a Geant4-specific raw geometry based on the GeoModel description. A tool called Geo2G4 has been developed for this purpose. This automatically translates the GeoModel description to Geant4 by navigating the GeoModel raw geometry graph. To make the resulting geometry memory-efficient a set of memory optimization techniques has been implemented, which are applied during the translation:

- If a logical volume is shared by leaf physical volumes in the GeoModel tree, the corresponding Geant4 logical volume will also be shared.
- If a physical volume is shared by several parents in the GeoModel tree the corresponding branch in Geant4 tree should also be shared.

The capability of translating GeoModel serial transformers into Geant4 parametrizations is provided through the G4STParameterisation class. This can be used directly in Geant4 applications for the purpose of creating G4 parametrizations without subclassing. G4STParameterisation objects perform volume parametrizations as in GeoModel, in particular with the use of TRANSFUNCTIONS.

3.5.2.7 Usage of the GeoModel Toolkit in ATLAS Software

The ATLAS GeoModel project started in summer 2002. Since then the geometries of all ATLAS detector subsystems have been described using GeoModel mechanisms. Reference [3-12] describes the way in which all of ATLAS is described in terms of GeoModel. The GeoModel descriptions of tracking detectors have been extensively used in simulation since ATLAS Data

Challenge 2. The tile calorimeter was also used in the simulation of data for the Rome Physics Workshop. The liquid argon calorimeter detector description using GeoModel has been completed recently, and will be used in future productions. Reference [3-13] describes the simulation of ATLAS and the operational experience in DC2. It is also planned to develop all subsequent versions of ATLAS detector (Initial Layout, Realistic Geometry etc.) using GeoModel and the Geometry Versioning System.

3.5.3 Generic Identification Scheme Connecting Events and Detector Description

The data coming from the ATLAS detectors require access to the appropriate "detector description" information in order to be used. For example, one needs to calibrate the individual channel responses, calculate their positions and correct for misalignment. This can be done in many ways, but ultimately relies on a key-lookup to match the readout data with its detector-description data.

The solution that has been chosen in ATLAS consists of an identification scheme that follows the logical hierarchical structure of the detector. For example, the SCT, silicon strip tracker, is composed of a barrel and two endcaps, and several layers of two-sided modules distributed in η and ϕ. Following this structure, the identifier specification of each silicon strip is:

InnerDetector/SCT/barrelorendcap/layer/phi_module/eta_module/side/strip

where each level is represented by a number. The hierarchical structure allows one to extract identifiers with the full or partial hierarchy – readout identifiers correspond to the full hierarchy. In ATLAS the following "three-level" model has been found to be useful:

Detector subsystem $\Rightarrow$ detector elements $\Rightarrow$ readout channels

where for the silicon tracker one would have SCT $\Rightarrow$ wafer (side) $\Rightarrow$ strip. The interest is in "projecting out" from this specification the identifiers for the detector subsystem, detector elements and readout channels. This three-level model is reflected in both the event and detector-description models. Identifiers are used as look-ups both within and between these models.

The software infrastructure to support this identification system consists of:

- an identifier dictionary that allows different identifier specifications to be captured and queried and the extraction of various forms of identifiers;
- various identifiers, e.g. compact and hash;
- identifier "helpers" which are specific to each detector system and simplify the interactions with the dictionary in terms of creation/manipulation of the identifiers.

The basic infrastructure has been in place since 1998, using identifiers in an expanded form (vector<short>). A complete specification of the identifiers for each detector system was completed in 2001, specified in an ATLAS software note [3-14]. In 2002 a migration to the use of an identifier dictionary was completed, which allowed the use of compact and hash identifiers that are more efficient in terms of space and look-up speed.

3.5.3.1 Identifiers, Identifier Dictionaries and Identifier Helpers

Two forms of identifiers are currently in use:

- Compact – 32 bit representation, the values of each level are bit-packed;
- Hash – 32 bit representation, transformation of a set of compact identifiers to numbers from 0 to N-1, allows constant-time table look-up.

The compact id contains the information of the identifier in bit-packed form. They can be used to extract the level information or as a binary look-up key. Hash identifiers have been introduced to improve upon the binary-lookup, reducing this to a constant-time look-up[1]. This is possible within a certain "context" or rather for a specific set of identifiers: the set of detector elements for a given sub-detector can be enumerated and each assigned its "hash" id that is simply a number from 0 to N-1. This hash id can be used in a table look-up.

The identifier dictionary formally defines a logical hierarchy and provides a name for each of the hierarchy levels. The identifier dictionary describes the set of non-overlapping regions of valid values for the hierarchy levels. The regions are divided into separate dictionaries for convenience, for example for each detector system, and are kept in separate files. This allows a representative for each detector system to maintain their part. These files can be versioned in the Hierarchical Versioning System (HVS) of the geometry database. As a first step, the file name is stored in the geometry database, and the contents of the file itself will eventually replace this.

A dictionary is first described as an XML file. Then a XML parser converts this dictionary into an IdDictDictionary object, which calculates the number of bits and allowed values for each identifier level. This allows efficient mask-and-shift operations to be performed for each level of the different dictionaries.

Identifier helper classes have been developed for each detector system to simplify the interactions with the dictionary. They localize to a single place in the software where identifiers may be created or decoded. Each helper is tailored to the specific detector identifiers, for example for decoding the specific levels. The helpers are initialized from their corresponding dictionary. The helpers transform a set of compact identifiers into their corresponding hash identifiers. The helpers can provide iterators over the set of identifiers. Finally, the helpers can provide fast access to neighbouring identifiers since this knowledge can be obtained from the dictionary, where one can specify, for example, that some hierarchy levels "wrap-around" in 2π for the cylindrical φ coordinate.

3.5.3.2 Impact on the ATLAS Data Model

The constant-time access of the hash identifiers is used in a variety of ways for each of the detector systems. For the calorimeters, the individual readout channels have identifier hashes so that the data (e.g. CaloCells) or their description (e.g. eta/phi) may be easily accessed. For the tracking detectors, a three-level container (IdentifiableContainer) has been developed:

$$\text{container} \Rightarrow \text{collection} \Rightarrow \text{object}$$

1. Performance measurements made on a 1 GHz Pentium show that unpacking a level takes 5–15 ns, depending upon the number of steps involved, and that a hash look-up varies from a few to 15 ns for vector sizes from 10k to 200k.

where the collection granularity corresponds to the detector elements, for example a silicon wafer, and the object may be a silicon strip/pixel. This is used for fast access to data in different contexts: for example in tracker pattern recognition or in the online software where a region of interest is transformed into a set of detector element identifier hashes, and the corresponding "collections" are decoded by the byte-stream converters.

3.6 Graphics and Event Display

3.6.1 Introduction

The need for graphical representation exists in all phases of the experiment. Graphics are used:

- for a precise visual description of the detector geometry;
- to provide rapid feedback for online monitoring purposes;
- as a development tool for reconstruction software;
- for displaying events to help in the analysis of both Monte Carlo and data, at the ESD/AOD level;
- to provide crisp and versatile event displays for publications.

The suite of graphics tools should meet the following requirements:

- provide full interactivity for displaying detectors and events at various levels of detail, ideally for all ATLAS detector systems;
- function in the ATLAS software framework; should also function as a stand-alone module for easy development and for specific tasks;
- be capable of providing 2D as well as 3D displays of single channels (hits, cells), tracks, clusters and the full detector. 2D is essential for simplicity and clarity, whereas 3D is particularly useful for the validation of the detector geometry and visualization of tracks in the complicated magnetic field configuration;
- display of both active and passive parts of the detector;
- provide easy access to evolving detector layout (e.g. incomplete detectors, missing channels etc.);
- provide adaptability for the various phases of ATLAS such as commissioning, cosmic and halo data, early collision data, routine data taking.

Additional requirements arise from needs encountered in performing specific tasks.

- In dealing with the detector geometry description, it is necessary to build complex geometrical objects and detect inconsistencies (clashes) in the layout. Using Boolean volumes is one way of achieving this.
- During the commissioning phase, the ATLAS graphics package must handle the evolution of the detector layout.
- For optimizing track reconstruction and clustering codes, one should be able to run these reconstruction codes within the ATLAS graphics package. It would be desirable to dis-

play important elements of these codes, such as the location of multiple scattering centres along tracks, as well as the magnetic field.

- At the analysis stage, the parameters of reconstructed tracks and/or clusters must be used to "follow" a particle throughout the detector.

Three separate graphics and event-display environments have been developed which address different aspects of these goals and requirements. They are described in the following sections.

3.6.2 Atlantis

Atlantis is an event-display program with the primary goal of the visual investigation and the understanding of the physics of complete events. In addition, it facilitates developing and debugging of reconstruction and analysis algorithms, and allows for debugging during detector commissioning.

The Atlantis program is based upon DALI [3-15], the event display used by the ALEPH experiment. It is written in Java and runs under different operating systems. Atlantis uses the experience obtained from DALI, which showed that data-driven 2D projections, while displaying a rudimentary description of the detector, provide excellent information for understanding events. The choice of 2D projections is driven by data and the detector layout. Sometimes non-linear projections (like ρ-z or fish-eye) are extremely helpful for track identification or extrapolation. New features for existing projections, as well as new projections, are added as a result of requests from the user community.

The colouring of data and detector outlines has been given special attention, aiming for an intuitive understanding of the important features of an event. Special colour maps, including grey-scale, are available for printing, so that the printed result is suitable for publication.

Atlantis allows the possibility of colouring hits and tracks by association, thus providing information on which hits are and are not used in the reconstruction as an aid to the debugging of reconstruction algorithms. A similar capability is available for colouring calorimeter information with respect to cluster and jet. By clicking on individual data elements, detailed information can be displayed, which again aids in debugging.

Atlantis reads both event and detector geometry information produced by Athena-based applications in XML-format. This can either be from a file which may contain either individual events or combinations of events, or using the XMLRPC network protocol. This technique allows for an independent development of Atlantis, using a well-defined interface. When obtaining information through the network protocol, it is possible to run Atlantis as an online event display on several machines simultaneously.

The detector geometry displayed in Atlantis is based on the primary GeoModel geometry which ensures that the displayed geometry is consistent with the event data, a capability that was used for the 2004 combined test beam (CTB) and will be used for ATLAS commissioning.

The network protocol allows fully bidirectional communication between Atlantis and Athena. This communication makes use of the interactive nature of Python-driven Athena applications, but moves the command prompt to the Atlantis program. Within Atlantis it will be possible to send Python commands to the Athena application, and subsequently display the event. Furthermore, Atlantis will have the functionality to save the XML-code of the modified event, and thus allows for a detailed visual comparison between different reconstruction settings (or ver-

sions) of the same event (even at a later time). Using this communication method, any future interactive functionality of Athena is automatically available through Atlantis, whilst Atlantis will still be available as a stand-alone program.

An example of the Atlantis event display is shown in Figure 3-2.

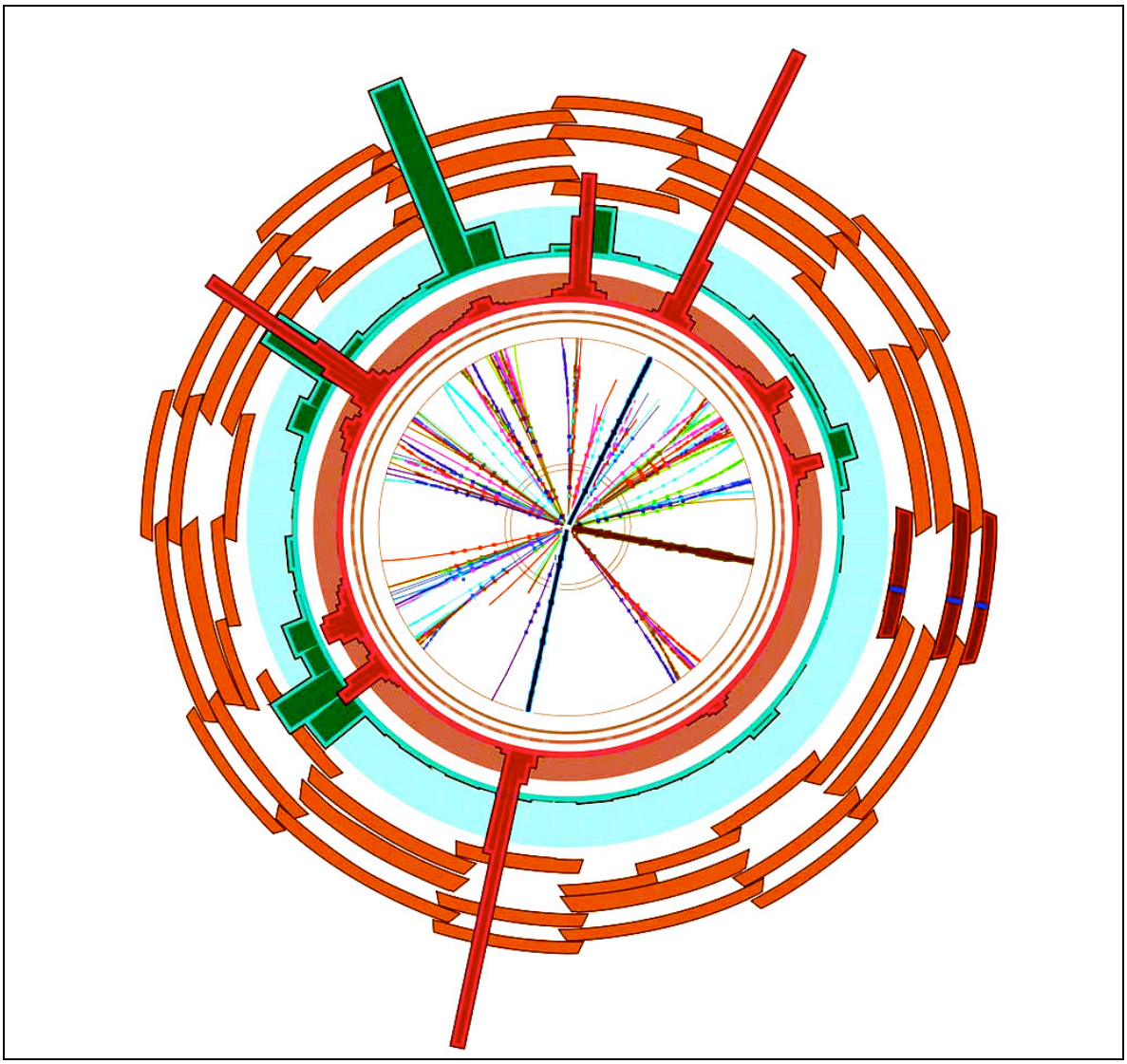

Figure 3-2 An example of the Atlantis Event Display

3.6.3 HEPVis/v-atlas

The visualization program HEPVis/v-atlas is the chief debugging tool for the ATLAS geometry described in Section 3.5.2. The system provides powerful interactive visual debugging of the geometry model. Running entirely within the Athena framework, it allows users to view the geometry at all levels of hierarchy and also to query the geometry. It is designed for extensibility, so that other visualization features such as track visualization and hit visualization are now possible, though it is designed primarily as a geometry debugger.

Since HEPVis/v-atlas runs within Athena, it has access to all transient data objects, including those associated with events (e.g. tracks, hits) and the detector geometry. Thus the visualization system accesses directly the same objects as are available within simulation, reconstruction, or analysis programs. HEPVis, the underlying library, has been a joint effort of several HEP institutes for a number of years. It's a set of HEP-specific extensions to the now open-source Open Inventor library. In addition to these libraries the package requires Motif.

An example of the HEPVis/v-atlas display is shown in Figure 3-3.

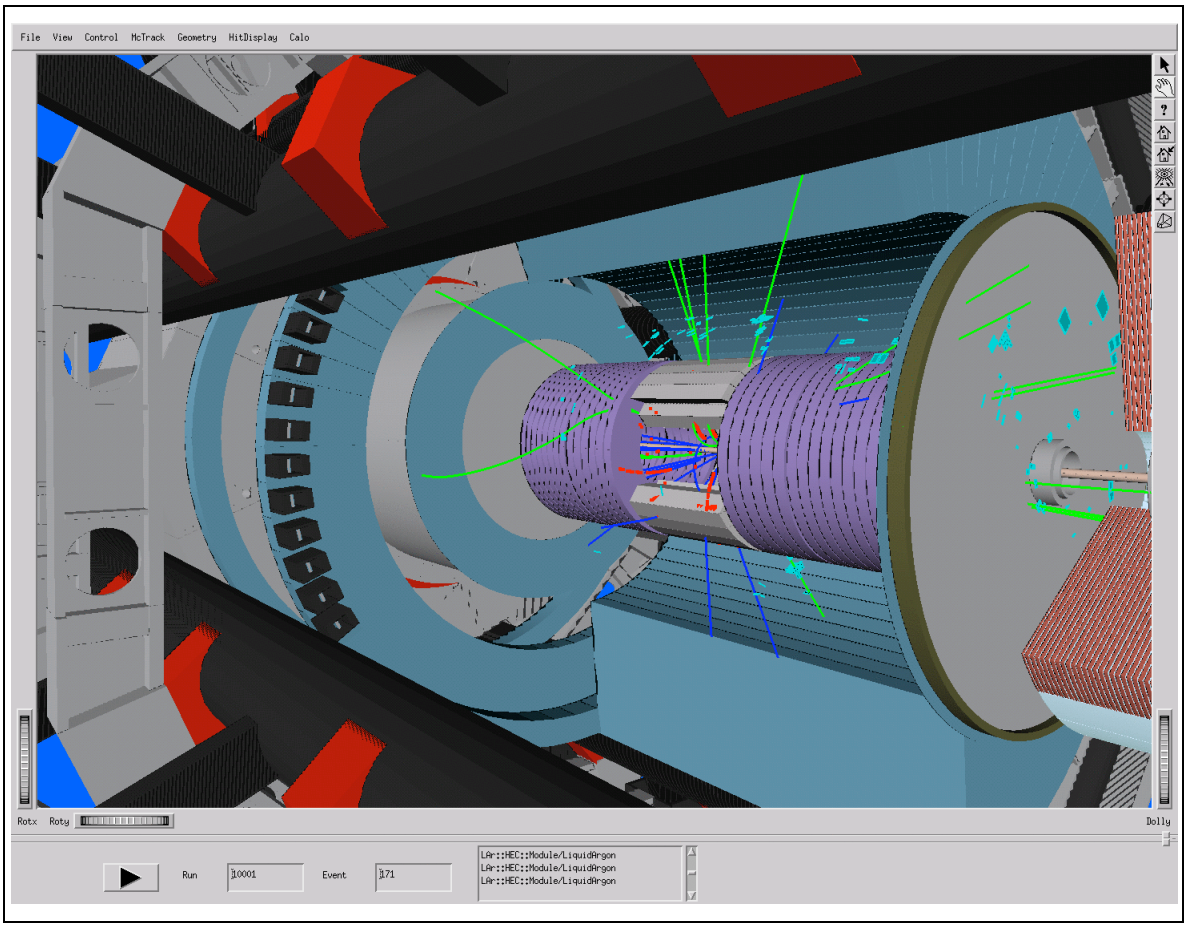

Figure 3-3 An example of the HEPVis/v-ATLAS Event Display

The program is broken down into modular components, each of which can be activated from a menu within the application GUI. Events can be selected from the event filtering fields and arrows in the display. Control points can enlarge or reduce the display area, the Event Filtering area, and the text window. Modularity is achieved by implementing each component as a "System" which translates StoreGate objects into Geometry (i.e. into Open Inventor), and a "Controller" which creates a menu tightly linked to the corresponding component. Each component places certain demands upon system resources, especially upon CPU, memory, and graphics acceleration. Users can have these demands under control by launching only the components they need.

HEPVis/v-atlas contains a browser for the geometry, virtual meter sticks for measurement purposes, a display of tracking detector simulated hits, a display of calorimeter hits and calibration hits, a display of Monte Carlo truth information, a data browser used to obtain a printout of cer-

tain banks, a mechanism for co-displaying VRML files with the detector geometry and/or hits, a display of tracking surfaces and track parameters at various stages of a Kalman Filter track fit, and a "template system" which can be copied and adapted to the needs of individual users or user communities as a starting point for specialized display. The system is designed to be easily extended so that developers can customize the display to their own needs, and to allow every major analysis to have a dedicated HEPVis/v-atlas display.

The display works best when run locally, i.e., on a laptop or desktop computer under a distributed release of the ATLAS software. The highest level of performance is obtained when a local replica of the geometry database is installed. HEPVis/v-atlas uses the object-oriented geometry modelling toolkit Open Inventor, which uses Open GL for rendering. The highest level of graphics performance is available on machines with a hardware accelerated graphics card.

3.6.4 Persint

The Persint program [3-16] is designed for the three-dimensional visual representation of objects and for the interfacing and access to a variety of independent applications, in a fully interactive way. It was designed particularly to deal with the complex requirements of the muon spectrometer and the magnetic field components. An example of the Persint display is shown in Figure 3-4.

The display of objects and the interactivity between the user and objects or applications is realized through the use of a graphical interface. Facilities are provided for the spatial navigation

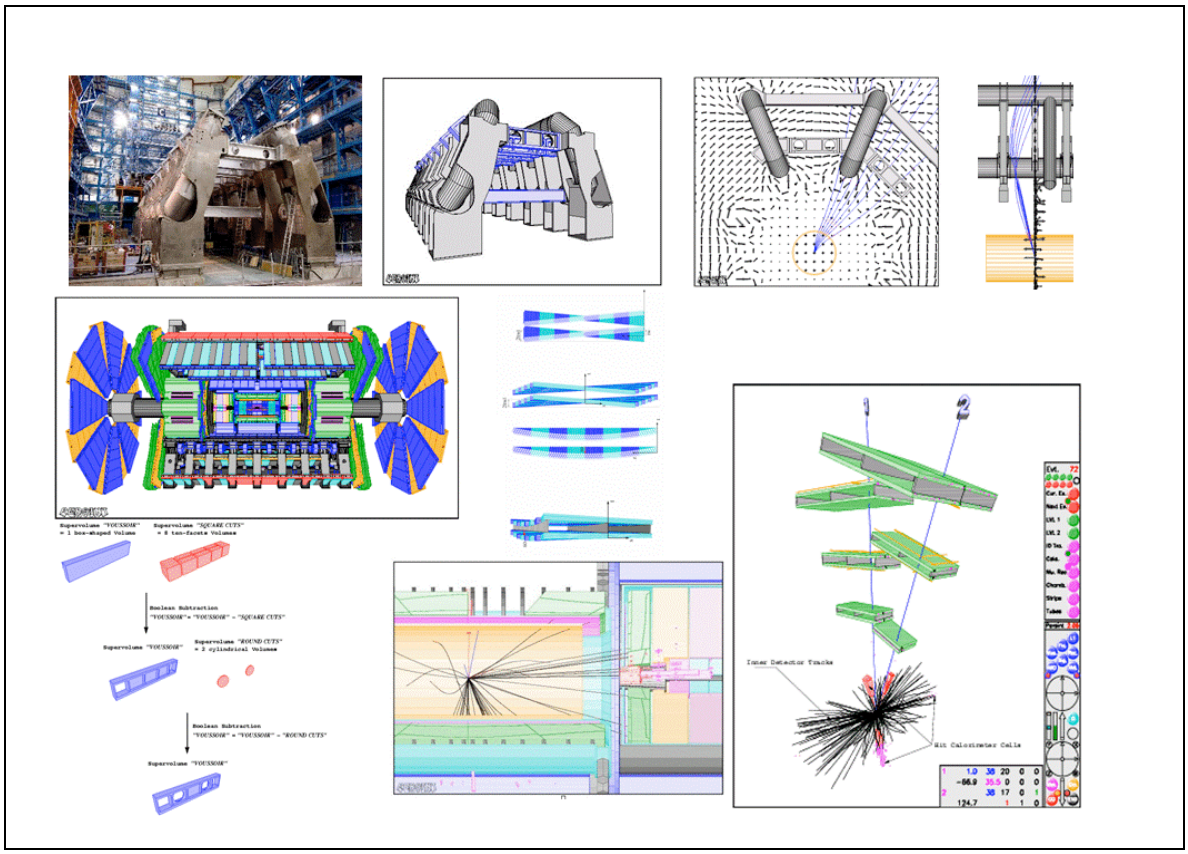

Figure 3-4 An example of the Persint Event Display

and the definition of the visualization properties, allowing the viewing and viewed points to be specified interactively, and to obtain the desired perspective. In parallel, applications may be launched through the use of dedicated interfaces, such as the interactive reconstruction and display of physics events. All information about detectors description (e.g. identifiers) and tracks (e.g. parameters, multiple scattering plane, energy loss, number of X0) can be displayed by a simple click at any time.

Persint works in stand-alone mode but could work as an Athena algorithm, although this has not yet been implemented.

Particular characteristics of Persint are:

- A precise visual description of the detector geometry. Persint allows a precise visualization of the detector geometry through AMDB, which is the source of the active detector part of Muon GeoModel and AGDD which is the detector dead matter part (see Ref. [3-17] for more details). The level of the geometry description for the muon spectrometer goes from the wire/tube to the full ATLAS detector. The pixels, silicon, TRT detectors and all calorimeters for the active part of the detector and all main dead materials (e.g. shielding, magnets) are also included in AMDB/AGDD but not at the same level of detail as the Muon spectrometer.

  The alignment of the Muon chambers is included in the detector visualization.

  In dealing with the detector geometry description, it is necessary to build complex geometrical objects and detect inconsistencies (clashes) in the layout. Using Boolean volumes is one way of achieving this. The XML-format of AGDD provides these features.

- Basic graphical features such as three-dimensional representation of objects in full volumes or wire frames, lighting-intensity effects on volume facets, computation of hidden faces, spatial navigation with real-time displacements, focal length adjustable from isometry to wide-angle, predefined projective views, interactivity with displayed objects (move, hide, change colour), etc.

- Advanced graphical features such as shape outlines, dynamical adjustment of the level of detail displayed on the scene, and the detection of volume interferences. The ability to draw object outlines plays a key role in debugging geometries and understanding complex HEP events. The ability to dynamically adjust the level of detail (small details viewed from a distance are not shown) improves considerably the performance of real-time navigation. Clashing volumes are automatically detected and their intersecting boundaries are highlighted, a useful feature in developing and debugging geometrical descriptions of complex systems such as the ATLAS Muon Spectrometer.

- The Level-1 Interface [3-18], an application allowing the interactive access to the Level-1 Muon Trigger configuration data files and the visual representation of the variety of objects involved in the trigger decision chain.

- Magnetic Field visualization. The four current sources that produce the magnetic field and the complex geometry of magnetic materials within ATLAS result in a very complex magnetic field configuration. Persint allows the visualization of the B-field vector at any point within the detector through its interface to the magnetic field map.

- A development tool for reconstruction software. Persint allows the visualization of any track (from the five parameters used to describe a track) under the influence of the magnetic field and shows the multiple scattering centres from any point within an ATLAS detector to any other location within ATLAS. This allows the complicated path of a track to

be understood and is used to draw tracks coming from the inner tracker. It provides interactively a precise estimate of the momentum resolution that can be achieved on the measurement of such tracks.

- High-quality vector PostScript output for publication-quality pictures.

Persint is able to run the reconstruction program Muonboy interactively. This is indispensable for the debugging of, and making improvements to, any reconstruction program in the muon spectrometer. Its 3D capability is essential because there is no analytical description of the track in the complicated ATLAS magnetic field.

3.7 Calibration and Alignment

The Computing Model aspects of calibration and alignment processing are discussed in detail in Section 2.5, including the different stages in the data processing chain where processing can be performed, and the various calibration streams. Apart from some calibration processing done directly in the sub-detector readout system (RODs), most processing will be done using specialized algorithms running in the Athena framework, either in the event filter, in dedicated Tier-0 resources, or offline. The responsibility for the corresponding software lies with the appropriate sub-detector communities, and detailed discussion of the approaches foreseen can be found in the ATLAS Physics Technical Design Report [3-19]. Many of these processes have been exercised (at least on a small scale) during the 2004 combined test beam.

Although primary responsibility for calibrating each sub-detector rests within the sub-detector itself, global calibration issues such as spatial and energy/momentum matching between the inner detector, calorimeters and muon system, determination of energy and momentum scales, and global coordinate systems will require ATLAS-wide coordination. The software for performing this level of global calibration will rely heavily on the combined reconstruction tools discussed in Section 3.9.2. Detailed strategies for this aspect of calibration and alignment have yet to be worked out, but will be developed as part of the computing system commissioning exercises in early 2006.

The results of calibration and alignment processing will be stored and distributed using the conditions database infrastructure, as discussed in Section 4.4. Calibration data generally appears as conditions data objects in the Athena transient detector store, using the same converter mechanisms as for event data. The link between Athena and the conditions database has been exercised for most sub-detectors for the first time in the combined test beam, using a MySQL conditions database implementation, and the upgrade of the infrastructure to access the new LCG COOL conditions database will be completed in time for the introduction of COOL for sub-detector commissioning in summer 2005.

3.7.1 Inner Detector

The SemiConductorTracker (SCT) and Pixel detectors each consist of a large number of detector elements, which are the fundamental units for the alignment. (Each detector element has several hundred readout channels.) The alignment strategy for the SCT and Pixels relies basically on two methods, which are being developed in parallel:

- A global chi-squared approach, in which all the correlations among the thousands of individual detector elements are handled simultaneously, and which implies the inversion of a single very large matrix;
- An iterative approach, in which the detector elements are aligned separately and the correlations are fed in indirectly.

The initial conditions for these methods will be extracted from survey data. The precision with which the detector elements have been mounted leads to an expected accuracy of the inter-module distance of 10 to 20 μm for the Pixel layers/disks, and of 50 to 100 μm for the SCT layers/disks.

For the SCT, an online monitoring of the cylinders and disks distortions will be performed by the FSI (Frequency-Scanned Interferometry) system, providing a run-time constraint to the track-based alignment.

The statistics needed to achieve the target precision (a few μm) is of the order of 10 million tracks. It is expected that the alignment will be performed every 24 hours or so, in running mode.

The TRT detector has 350 000 straws whose position and calibration constants need to be determined. Initial survey, together with test-beam and cosmic-ray calibration, will provide a starting point with a spatial precision of about 200 μm.

All further calibration and alignment will have to use the collider data themselves. An iterative procedure is carried out where each iteration performs some of the tasks in the following sequence:

- The position correction and t_0 for the channel is fitted to the average raw time as a function of track impact position. Internal TRT tracks are used for this.
- The corrections are fitted to rigid transforms of barrel modules or endcap wheel sectors relative to each other. The remaining corrections are fitted to a second degree polynomial in the second coordinate.
- The average track distance from the corrected wire is fitted to a constrained polynomial as a function of drift time.
- Using external or global tracks, the rigid transform of an entire barrel or endcap phi-sector is determined relative to the silicon system.

There are in principle three alignment and six calibration constants per lowest level detector element (the straw), including individual coordinate resolutions and high-threshold hit efficiencies. However, they are stored in structures that may flexibly share calibration values among larger or smaller groups of straws.

The statistics needed for obtaining a 10 μm alignment precision can be extrapolated from test-beam experience (25 000 tracks exposing 230 straws), which yields of the order of 10 million tracks with $p_T > 2$ GeV. A dedicated stream running before the Tier-0 reconstruction is foreseen to handle the TRT calibration task.

3.7.2 Liquid Argon and Tile Calorimeters

The goals of the ATLAS calorimeters are the accurate measurement of the energy of electrons, photons and jets, the measurement of missing transverse momentum, and particle and jet identification.

A resolution of better than $\Delta E/E = 10\%/\sqrt{E} \oplus 1\%$ (E in GeV) is required for EM calorimetry to guarantee 1% resolution on a light Higgs mass in the $H \rightarrow \gamma\gamma$ or $H \rightarrow 4e$ decay channels. Requirements on the energy resolution for the hadronic calorimeter are $\Delta E/E = 50\%/\sqrt{E} \oplus 3\%$ for $|\eta| < 3$ and $\Delta E/E = 100\%/\sqrt{E} \oplus 10\%$ for $3 < |\eta| < 5$. Earlier studies documented in the ATLAS Physics TDR have indicated that such resolutions are adequate for the tasks of providing jet reconstruction as well as missing p_T measurements for the physical process of interest.

An energy-scale precision of about 0.1% is desirable for EM calorimetry for, for example, W mass measurement, Higgs and various SUSY searches. For heavy vector boson searches the dynamic range up to 5–6 TeV with 2% non-linearity is required, while below 300 GeV a linearity in the response of better than 0.5% is vital for the precision measurement of a light Higgs or top mass.

In order to meet these physics requirements a careful calibration of the calorimeter is performed in three steps:

- Electronic calibration.
- Electromagnetic scale calibration.
- Calibration of the reconstructed objects (clusters, particles, jets).

In ATLAS the energy deposited in the calorimeter is calculated using Optimal Filter (OF) method as:

$$E = C \times \sum_{i=1,n} f_i \times (A_i - P)$$

where A_i are measurements[1] of the calorimeter response with 40 MHz frequency, f_i are OF coefficients, C is the overall scale factor to convert ADC counts to energy, and P is the electronic pedestal. A similar formula is used to calculate signal time.

During special electronic calibration runs the scale factors to convert from ADC counts to nanoamperes (or picocoulombs in case of TileCal) as well as all OF coefficients are obtained for all 2×10^5 channels. OF coefficients depend on the signal peaking time and usually 25 different sets are calculated in 1 ns steps. Various correlation matrices are also calculated and, in total, approximately 10^3 parameters per channel need to be stored in the Conditions Database. The complete set of calibration coefficients in the database is expected to be updated every few days at ATLAS start-up and approximately once per month once stable running conditions have been achieved.

The EM scale calibration provides nA/GeV (or pC/GeV) conversion factors for all calorimeter channels. Initial calibration has already been performed over the last 10 years at a variety of beam tests and with detailed Monte Carlo studies. This calibration will be refined *in situ* during real data-taking. Both the OF coefficients and EM scale factors will be loaded from the Condi-

1. Summed over five measurements for the liquid argon calorimeter and seven for the tile calorimeter.

tions DB to Digital Signal Processors (DSPs) at the start of every data acquisition run, so that the output from the DSP will be the energy calibrated at the EM scale.

The final calibration step, a calibration of final reconstructed objects, tries to compensate as much as possible for various inhomogeneities in the calorimeter and for differences in the electron and pion response. Once again, beam test measurements, Monte Carlo studies and in-situ calibration will be used to obtain the calibration coefficients.

For *in situ* calibration a large data sample (approximately one month of data taking) needs to be processed in order to achieve good uniformity and resolution. Nevertheless, fast reconstruction of special calibration streams, e.g. $Z \rightarrow e^+e^-$, should make it possible to achieve a global constant term of about 0.7% from a few days of data taking and to provide fast feedback to the online system. The expected rate of $Z \rightarrow e^+e^-$ is about 1 Hz at low luminosity which means that prompt reconstruction should be feasible at the same rate.

3.7.3 Muon Spectrometer

The muon spectrometer is designed for a relative momentum resolution, $\delta p_T/p_T < 1.0 \times 10^{-4}$, for $p_T > 300$ GeV. In order to achieve this resolution by a three-point measurement, with the size and bending power of the ATLAS toroids, each point must be measured with an accuracy better than 50 μm. Monitored Drift Tube (MDT) chambers, built with high mechanical accuracy (30 μm), are used for this purpose over most of the solid angle covered by the spectrometer. However, in order to meet the design accuracy, the chamber deformations and positions must be constantly monitored by means of optical alignment systems (hence the name MDT), and the conversion from measured times to drift radii should be carefully calibrated in order to introduce an additional uncertainty no larger than 30 μm [3-20].

For wire-position monitoring, 7000 optical sensors are mounted in the barrel and 10 000 in the endcap, each sensor providing four measurements. This information is used to fit the geometry parameters describing the position (6 parameters) and the distortions (8 parameters) for each chamber. In total 9100 parameters are fitted in the barrel and 7700 in the endcap. In addition to the optical information, straight muon tracks previously selected by the trigger will be used and integrated into the alignment fit. Alignment information will be updated every 20 minutes.

The MDT calibration provides the time offset (t_0) to better than 1 ns from the drift-time distribution for each of the 370 000 tubes and extracts the strongly non-linear space–time relationship through an auto-calibration procedure, making use of the tracks reconstructed in the MDT chambers. The space–time relationship depends on gas-drift properties and is sensitive to temperature and magnetic field variations: it is then computed for groups of tubes sharing similar working conditions. The total number of space–time relationships needed to describe the whole MDT system at a given time is of the order of a few thousand.

As discussed in detail in Ref. [3-21] the t_0 and space–time relationship computation requires large muon samples. A minimum number of 20 000 hits per tube are needed for a good t_0 computation. A rough stability check of t_0, to be repeated daily, requires at least 5 000 hits/tube. This number corresponds to about 100M muon tracks per day, a sample adequate also for the space–time relationship computation.

The means to collect the required statistics (through dedicated runs or through a continuous calibration stream running at few kHz) are currently under discussion. However, it should be not-

ed that only MDT and trigger detector hits are needed, so the corresponding data size should be limited to about 1 kB/event.

The calibration is an iterative process but the most demanding operations in terms of CPU and data handling (decoding, access to conditions database, access to geometry, and pattern recognition) need only be performed once to obtain and store the drift times for each hit in a track segment. Currently, the best estimate of the time needed for the initial iteration of a muon track is 100 ms on a 2.5 GHz processor. As subsequent iterations just modify the space–time relationship for the hits and refit the track segments, they are much faster. The processing time will at most double after all iterations. Therefore, the processing of a 100M event data set will require approximately 6000 CPU-hours.

Validation of the calibration is needed before releasing the constants to the conditions database and will require additional CPU time. To provide the calibration constants within 24 hours of data collection, a minimum of 300 2.5 GHz processors is needed for all the calibration tasks listed above [3-22]. These processors could be part of Tier-0, or the CERN analysis facility, or could be located remotely. The pro's and con's of the two solutions are under investigation. Remote farms would make additional requirements on both network connections (but the necessary bandwidth is only of the order of 10 MB/s) and on database distribution (up-to-date DCS information is needed to perform the calibration).

3.8 Simulation

3.8.1 The Simulation Data Flow

Figure 3-5 shows a simplified view of the processing stages in the simulation data flow. Input for simulation comes from event generators after a particle filtering stage. Data objects representing Monte Carlo truth information from the generators are read by simulation and processed. Hits produced by the simulation can be directly processed by the digitization algorithm and transformed into Raw Data Objects (RDOs). Alternatively they can be sent first to the pile-up algorithm and then passed to the digitization stage.

RDOs produced by the simulation data-flow pipeline are used directly by the reconstruction processing pipeline described in Section 3.9. Thus the simulation and reconstruction pipelines are coupled together by the RDOs which act as the output from the simulation pipeline and the input to the reconstruction pipeline. However, the ATLAS TDAQ produces byte-stream files and, in order to reproduce this, an optional final stage can be added to the simulation processing chain in order to generate these files from the RDO files. In this case, the initial stage in the reconstruction pipeline first converts the byte-stream information into RDO objects which are then used for subsequent reconstruction processing.

Any Monte Carlo truth information is removed in the conversion from RDOs to byte-stream format such that byte-stream files produced by the simulation pipeline are an accurate representation of the byte-stream files coming from the ATLAS TDAQ. In the context of the High-Level Trigger (HLT), the conversion from byte-stream representation to RDOs is in some cases replaced by a direct conversion to objects (PrepRawData) otherwise created by the next stage in the reconstruction pipeline, but this is a performance optimization only.

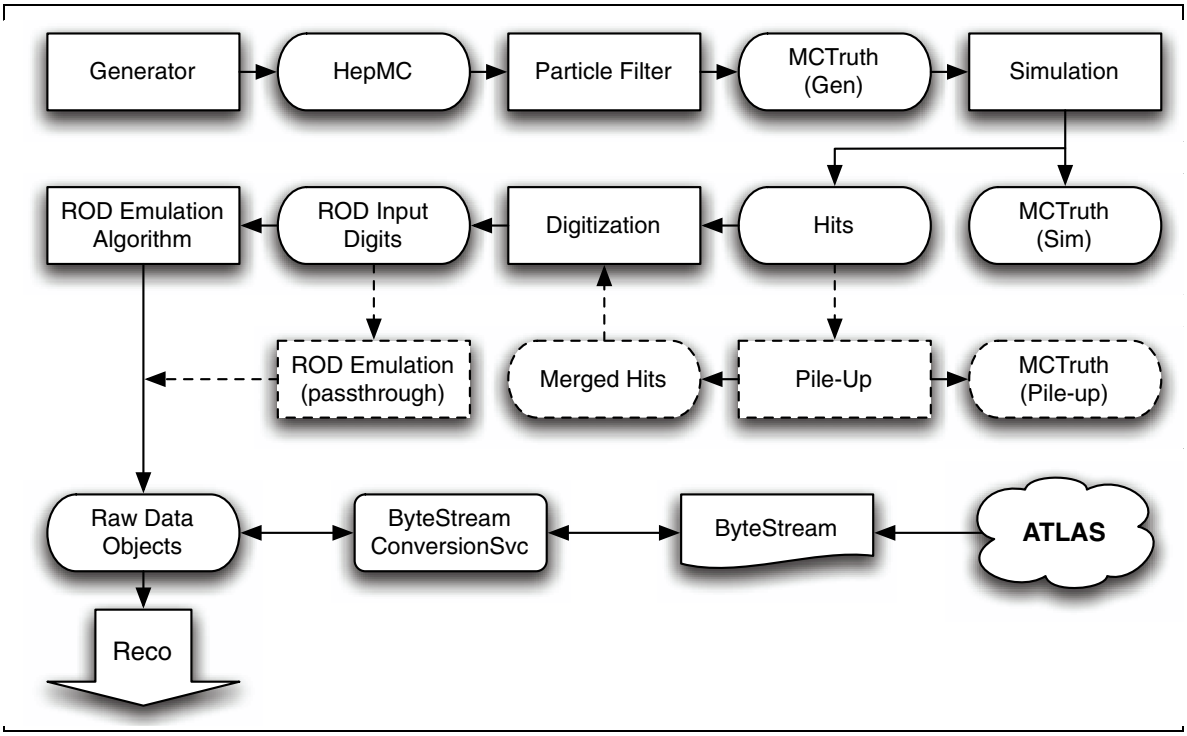

Figure 3-5 The simulation data flow. Rectangles represent processing stages and rounded rectangles represent objects within the event data model. Pile-up and ROD emulation are optional processing stages.

The stages in the simulation data-flow pipeline are described in more detail in the following sections. In addition to the full simulation framework, ATLAS has implemented a fast simulation framework that substantially reduces the processing requirements in order to allow larger samples of events to be processed rapidly, albeit with reduced precision. Both these frameworks are described below.

3.8.2 Generators

Event generators are indispensable as tools for the modelling of the complex physics processes that lead to the production of hundreds of particles per event at LHC energies. Generators are used to set detector requirements, to formulate analysis strategies, or to calculate acceptance corrections. They also illustrate uncertainties in the physics modelling.

Generators model the physics of hard processes, initial- and final-state radiation, multiple interactions and beam remnants, hadronization and decays, and how these pieces come together.

The individual generators are run from inside Athena and their output is converted into a common format by mapping into HepMC. A container of these is placed into the transient event store under StoreGate and can be made persistent. The event is presented for downstream use by simulation, for example by G4ATLAS simulation (using Geant4) or the Atlfast simulation. These downstream clients are thereby shielded from the inner details of the various event generators.

Each available generator has separate documentation describing its use. Simple Filtering Algorithms are provided, as well as an example of how to access the events and histogram the data.

The current list of supported Generators includes Herwig, Pythia, Isajet, Hijing, AcerMC, CompHep, AlpGen, Tauola, Photos, Phojet and ParticleGenerator. Some utility classes to enable filtering of events and facilitate handling of Monte Carlo Truth are also provided.

The code is organized into the following packages:

- GeneratorModules contains the base classes from which the specific Generators inherit.
- Pythia_i contains the code for the Pythia interface and the Algorithm to load Pythia.
- Herwig_i contains the code for the Herwig interface and the Algorithm to load Herwig.
- Isajet_i contains the code for the Isajet interface and the Algorithm to load Isajet.
- Hijing_i contains the code for the Hijing interface and the Algorithm to load Hijing.
- Tauola_i contains the code for the Tauola interface and the Algorithm to load Tauola.
- Photos_i contains the code for the Photos interface and the Algorithm to load Photos.
- AlpGen_i contains the code for the AlpGen interface and the Algorithm to load AlpGen.
- Phojet_i contains the code for the Phojet interface and the Algorithm to load Phojet.
- ParticleGenerator contains the code for the ParticleGenerator interface and the Algorithm to load ParticleGenerator.
- CompHep_i contains the code for the CompHep interface and the Algorithm to load it.
- AcerMC_i contains the code for the AcerMC interface and the Algorithm to load it.
- MCNLO_i contains the code for the MCNLO interface.
- GeneratorUtils contains some utility routines.
- GeneratorFilters contains some examples of how to filter events.
- GenzModule provides the ability to read events made by the G3 Simulation and pass them into Athena in a uniform manner; for example, so they can be used by Atlfast.
- GeneratorObjectsRoot is a package that outputs and inputs the events in Root I/O format.
- GeneratorObjects sets up the containers which will hold the events in a collection of HepMC events. McEvent also sets up the serializers for ROOT.
- McEventSelector is responsible for assigning run numbers and providing the hooks to Gaudi-Interfaces, it uses EventAthena.
- GenAnalysisTools is a set of algorithms for Generator analysis, it is used mainly by reconstruction. It contains three packages:
    - CBNT_Truth, used for the truth part of the CBNT combined n-tuple.
    - TruthExamples. This has examples to show histogramming and listing of generated events.
    - TruthHelper, containing helper classes for extracting stable particles, for example.
- PythiaB, the specific version of Pythia used by the B-physics group.

Only the interface code is held in the ATLAS CVS repository. The code supplied by the external authors is built and maintained either by the LCG Genser project [3-23] or by ATLAS members in an external area. The long-term goal is to have it all done by Genser. The HepMC package was an ATLAS product that is now being maintained by CLHEP.

Most of the event generators are in FORTRAN; the ATLAS interfaces are in C++. The migration to C++ has been slow, and currently Sherpa is the only fully functional C++ generator. It has been tested and evaluated by ATLAS in the production for the Rome Physics Workshop. It is run as a stand-alone product that writes data files that are imported to Athena using an interface algorithm; this is currently being re-evaluated.

Two other C++ generators, Herwig++ and Pythia7, are not yet fully functional. Herwig++ cannot yet be used for processes with hadrons in the initial state and Pythia7 lacks the functionality of the FORTRAN version. Given the time needed for development, deployment, testing and validation of a new generator, it is clear that FORTRAN support will be required for some considerable time after LHC data is available.

3.8.3 Fast Simulation (Atlfast)

3.8.3.1 Overview

The ATLAS fast simulation program (Atlfast) simulates ATLAS physics events, including the effects due to detector response and the software reconstruction chain. The input to the program is the collection of four-vectors for a physics event, usually provided by a physics event generator. Atlfast examines the event record for stable particles.

Four-vectors corresponding to electrons, photons and muons are passed to the appropriate smearing function, and the resulting four-vectors are output for use by downstream physics analysis. The calorimeter response to the event is calculated by summing the transverse energy deposits of all particles.

Smearing and jet finding algorithms are applied to the energy deposits, and the resulting jet objects are output for further physics analysis.

Other quantities calculated by Atlfast are track helix parameters and global event quantities such as total E_t and missing momentum.

3.8.3.2 Current Status

The current version of Atlfast is an adaptation of the original stand-alone FORTRAN program [3-24]. It has been rewritten in C++, and the structure has been heavily modified to adapt to run within the Athena framework. The physics results are found to be extremely close to the original FORTRAN version.

3.8.3.3 Goals for the Eventual Turn-On System

A new development is work on a comparator to compare the fast simulation to the full simulation. The idea is to run the full simulation and reconstruction, fit various distributions, and feed the information back into Atlfast, primarily through the smearing functions. The process is to be iterated until an acceptable level of agreement has been obtained. An important aspect of this work will be the documentation and control of the potentially many parameter sets that will become available to Atlfast. Some development on Atlfast itself will be necessary to provide the

infrastructure of passing the parameter sets to Atlfast, particularly as many of the original parameter sets are at present explicitly in the code.

Further work is expected on the FastShower library that simulates energy deposits in the towers of the ATLAS calorimeters, where the modelling of the deposition process includes two compartments in depth (electromagnetic and hadronic), and transverse shower spread. The correlation of energy deposits among neighbouring towers is included. FastShower is already included in Atlfast, but further effort to validate the calculations, and to deal with calibration issues, is required. The energy flow algorithms require knowledge of the expected total calorimeter energy deposit probability distributions as a function of particle type (e, π, μ) and of kinematic properties ϕ, η, and p_T. It is vitally important that this information be at the electromagnetic scale, to avoid any dependence on the hadronic calibration scheme. Further validation and refinement will be obtained using the full simulation comparator.

3.8.4 ATLAS Geant4 Simulation (G4ATLAS)

The ATLAS detector simulation programs have been heavily based on the Geant3 simulation package and infrastructure since the inception of the experiment. These programs were used in the preparation for the ATLAS Letter of Intent [3-25], in detector optimization studies, for the various sub-detector Technical Design Reports, and they were stress-tested in Phase 1 (i.e. the event-generation and simulation phase) of Data Challenge 1 (DC1), which was run during summer 2002. Geant3 has been a powerful, reliable and successful tool within ATLAS for about ten years.

With the development and implementation of the Geant4 (G4) toolkit [3-11][3-26], starting from the year 2000, ATLAS prepared for moving its simulation suite to the object-oriented (OO) paradigm. Geant3 and Geant4 were run alongside each other for a while in order to validate the new suite against the previous one. The switch-over [3-27] finally happened in 2003, in the early preparation phase of the second Data Challenge (DC2). Since then, Geant4 has become the main simulation engine of ATLAS, and all new developments have been carried out in the new environment. Geant3 support within ATLAS is being phased out during 2005.

The Geant4 toolkit provides both a framework and the necessary functionality for running detector simulation in particle physics and other applications. The functionalities provided include optimized solutions for geometry description and navigation through the geometry, the propagation of particles through detectors, the description of materials, the modelling of physics processes (e.g. a huge effort has been invested in recent years into the development and improvement of hadronic-physics models), visualization, and many more. A basic concept is that of *Sensitive Detectors*, which allow for the definition of active detector elements, perform corresponding actions within them, and write out *hits* (which may carry information like position, energy deposit, identifier of the active element, etc.). Geant4 is part of the common LCG application software project, and its development is pursued as a world-wide effort, coordinated by a strong development team from CERN.

Development activities to make use of Geant4 functionality within the ATLAS-specific set-up and software environment started in 2000, taking into account ATLAS-specific requirements. These provide tailored packages for handling geometry, kinematics, materials, physics, fields, sensitive detectors, run-specific issues and visualization, etc. These activities culminated in 2003 with the Geant4 simulation being embedded in Athena. This migration to Athena was also done for the detector simulation packages which had been developed in detail in the standalone environment.

In general, Geant4-based detector simulation programs (G4ATLAS) are based on criteria like dynamic loading and action-on-demand, and all user-requested functionality has been added by means of plug-in modules. Since 2003, extended common functionality and new developments have been implemented only in the Athena-based version; examples are updates on physics processes (e.g. transition radiation process for TRT simulation), the implementation of Monte-Carlo truth, simulation of the ATLAS combined test-beam set-up, the usage of POOL to write persistent output, etc. A particularly important new feature is the building of the Geant4 geometry tree from the one implemented in the ATLAS Detector Description package *GeoModel* (described in Section 3.5) using the *Geo2G4* conversion package. This procedure has the clear advantage of avoiding duplication of efforts and extra work involved in maintaining and synchronizing two different detector geometry versions, one for the simulation, the other for reconstruction.

A concise, but more detailed overview on the status and functionality of G4ATLAS can be found in Ref. [3-28].

Since 2001 a rather extensive physics validation programme has been under way to test the physics models implemented in Geant4, to ensure, through comparison with test-beam results where available, that Geant4 simulation meets the expected precision targets, and to provide feedback to the Geant4 development team. In almost all cases comparison with experimental data from beam tests gives very good agreement, normally at the level of ~1% or better in predictive power.

In addition to the physics performance, the validation and optimization of the G4ATLAS runtime performance, in particular of its CPU and memory requirements, has also been given high priority. Intensive validation tests started in autumn 2003, soon after a fully-fledged G4ATLAS simulation had become available within Athena. This effort produced a stable, robust and high-performance version, which was then run in DC2 simulation half a year later. In DC2, which was the first data challenge entirely based on Geant4, more than 12 million full physics events in more than 100k jobs were successfully simulated over four months in a world-wide, distributed way, with only one reported crash due to Geant4. For example, at NorduGrid a total sample of more than 3.5 million events (including 1 million full $Z \rightarrow e^+e^-$ events) was processed in about 35k jobs without a single reported failure. The validation programme has continued after DC2; a summary of the activities between autumn 2003 and the end of 2004 can be found in Ref. [3-29].

The G4ATLAS developers team is constantly working on improvements to the package and on extensions to its functionality, taking into account user requirements. Recent examples are the "pythonization" of the G4ATLAS user interface, which replaces the Geant4 user interface based on macro files. The power of Python, its flexibility, and the possibility of running and configuring a job interactively are clear advantages, in particular when many different configurations have to be handled, as in combined test-beam simulation. Another recent example is the implementation of a generic utility to enable and facilitate the usage of parametrized detector responses, e.g. for the liquid argon calorimeter, which can be used to speed up simulation jobs.

3.8.4.1 Simulation Performance and Prospects

The ATLAS Computing Model assumes that simulation requires approximately 100 kSI2k-sec per event processing time. Recent measurements on a representative set of physics events range between about 250 kSI2k-sec per event for minimum-bias events, to 850 kSI2k-sec per event for SUSY/SUGRA events. However, several optimizations are under active development:

- The parametrization of the detector response, particularly for the calorimeters. The impact of this on the physics performance will need significant study and is expected to be selectively applied only to particular physics channels or pseudorapidity regions.
- A re-evaluation of the Geant4 cut optimization. A more detailed study of the present cuts indicate that they are too tight, and can be significantly loosened without significantly impacting the physics performance.
- The application of more flexible pseudorapidity acceptance cuts, depending on the event type, since not all physics studies need the full detector coverage. The present performance numbers correspond to the full detector acceptance (pseudorapidity cut $|\eta| < 6$) for all physics channels.
- Continued optimization of the simulation code.

These optimization developments are expected to significantly improve the CPU performance such that the performance goal will be met.

3.8.5 Pile-up

G4ATLAS produces hits as output, which are a record of the real interactions of particles in the detector. At higher machine luminosities, however, multiple interactions can occur at each beam crossing (typically one signal event with multiple minimum-bias background events), and in addition other backgrounds (e.g. cavern background) need to be taken into account. As seen in Figure 3-5, pile-up (i.e. the overlaying of signal and background events) is an optional processing stage in the simulation processing pipeline.

For DC2, where pile-up at a luminosity of 10^{34} was processed, over 800 background events were overlaid over one event of the original physics stream. The main requirement was that the digitization algorithms should run unchanged. Many optimizations of data structures and access patterns were necessary in order to allow pile-up to run on a standard processing node, as original memory consumption was excessive (as high as 3 GB). Memory requirements for DC2 at a luminosity of 10^{34} (which excluded cavern background) were less than 1 GB. The inclusion of cavern background and changes in the timing window (to provide a more realistic description for the muon spectrometer) for the Rome Physics Workshop have increased the memory requirement to 1 GB at a luminosity of 2×10^{33}, so further development will be needed in order to keep below this memory threshold at higher luminosities.

The Athena-based pile-up application manages multiple input streams. Random permutations of events are selected from a circular buffer of minimum-bias events. Since the various sub-detectors have different data integration times, they require individual cache retention policies. By using a two-dimensional detector and time-dependent event caching policy, memory utilization has been significantly reduced.

Pile-up is an excellent mechanism to stress-test the architecture. Small problems that would normally pass unnoticed may get enormously magnified and become visible far sooner. It is also an excellent tool to expose memory leaks, as they might become magnified by several orders of magnitude (depending on the luminosity).

3.8.6 Digitization

The hits produced either directly by G4ATLAS, or from the merging of pile-up events, need to be translated into the output actually produced by the ATLAS detectors. The propagation of charges (as in the tracking detectors and the liquid argon calorimeter) or light (as in the case of tile calorimeter) into the active media has to be considered as well as the response of the readout electronics. Unlike the previous steps in the simulation chain, this is a very detector-specific task, and the expertise of people building and testing each of the sub-detectors is essential. The final output of the digitization step are Raw Data Objects (RDOs) that should resemble the real detector data. In addition to RDOs, Simulation Data Objects (SDOs) are created to save some simulation information that may be useful to the downstream user. The navigation between SDOs and RDOs is achieved by using identifiers[1], and SDOs are otherwise completely decoupled from RDOs to avoid any dependency on simulation that will not be there when real data is reconstructed.

To implement the modular organization of digitization, a package is created for each of the detector subsystems and a single point of contact is available for each package. Design and operating conditions (like magnetic field or voltage) of the detectors are set using job-option parameters or taken from the condition or detector description database. Digitization operates locally at the level of each sub-detector (e.g. a pixel module or a calorimeter cell) and the same code can be used in the context of the full ATLAS simulation, or a test beam or any other test. It is of key importance that digitization is tuned by comparing the RDO output to real data in system tests to produce a realistic tuning of the detector response.

A package is provided to put together and coordinate the sub-detector efforts. Only Python code is available in this digitization package to steer the code from the detectors. Detector flags and global flags are used according to the ATLAS policy to switch on/off detectors and tasks.

The digitization package was successfully used for DC2 and for subsequent physics productions as a separate step to be run after G4ATLAS. Although digitization was used only on Geant4 simulated events, there is no dependency between the ATLAS digitization package and Geant4, with the decoupling done at the Event Data Model (EDM) level by not having a dependency between the hits and Geant4. An automated test for digitization is also maintained in the nightly builds, and is used to spot problems that may arise in any of the sub-detector packages to stimulate rapid corrective action.

3.9 Event Selection and Reconstruction

3.9.1 Introduction

The ATLAS detector will produce approximately 3 PB of raw data per year, a vast amount of information which prohibits the simple distribution to worldwide collaborators. To enable physicists to analyse the data at remote sites, several different types of datasets, corresponding to different stages of reconstruction, are produced. Thus the following datasets are available:

1. Identifiers are described in Section 3.5.3

- *Byte-stream Data* which is a persistent presentation of the event data flowing from the HLT.
- *Raw Data Object Data* (RDO) which is a C++ object representation of the byte-stream information.
- *Event Summary Data* (ESD) which contains the detailed output of the detector reconstruction and is produced from the raw data. It contains sufficient information to allow particle identification, track re-fitting, jet calibration etc. thus allowing for the rapid tuning of reconstruction algorithms and calibrations. The target size for the ESD is 500 kB per event.
- *Analysis Object Data* (AOD) which is a summary of the reconstructed event, and contains sufficient information for common analyses. Several tailor-made streams of AODs are foreseen for the different needs of the physics community. The AOD can be produced from the ESD and thus makes it unnecessary in general to navigate back and process the raw data, adding significant cost and time benefits. The target size for the AOD is 100 kB per event.

Inevitably, there will be some overlap between the different reconstruction realms; for example, some objects will exist in both AOD and ESD. There will also be "tags" on each event, summarizing some general features of the event, and allowing in particular selection of and rapid access to a subset of events. The target size for the tags is 1 kB per event.

The reconstruction processing pipeline can be decomposed into several stages as summarized in Figure 3-6. Primary stages are:

- Detector and combined reconstruction (henceforth "Reconstruction"): This includes the reconstruction of the tracking and calorimetry detectors and the first steps in particle identification. The output is stored and defines the content of the ESD.
- Analysis preparation. This step includes the reconstruction of complex objects, for example the b-tagging object JetTag, and reduces the information to an acceptable size for wide distribution. The output defines the AOD content. Furthermore the event tags are created from the AOD in an additional step. For the physics analysis of the data on the AOD level, a variety of tools have been developed to encapsulate the complexity of the data objects to the user. These are described in Section 3.10, "Physics Analysis Tools".

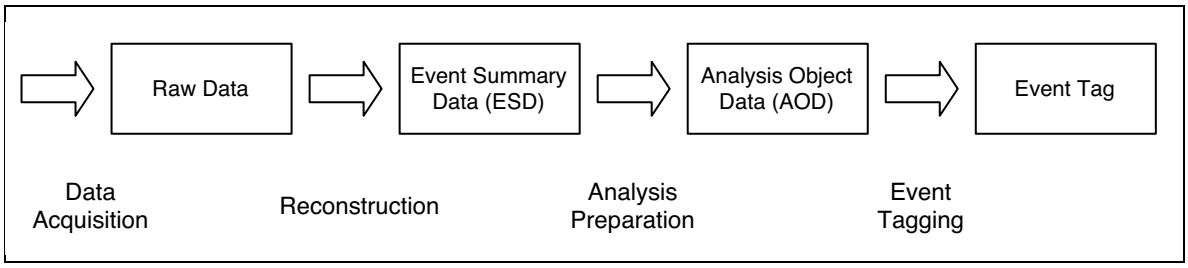

Figure 3-6 The reconstruction processing pipeline

3.9.2 Reconstruction

The role of reconstruction is to derive from the stored raw data the relatively few particle parameters and auxiliary information necessary for physics analysis: photons, electrons, muons, tau-leptons, K^0s, jets, missing transverse energy, primary vertex. Information from all detectors is combined so that the four-momentum reconstruction is optimal for the full momentum

range, full rapidity range and any luminosity, and so that particles are identified with the least background, with the understanding that the optimum between efficiency and background rejection can be analysis-dependent.

A typical reconstruction algorithm takes one or more collections as input, calls a set of modular tools, and outputs typically one collection of reconstructed objects. Common tools are shared between tracking detectors on one side (inner detector and muon spectrometer) and calorimeters on the other side (liquid argon electromagnetic calorimeter, hadronic endcap and forward calorimeter, and tile hadronic detector). Reconstruction tools can share interfaces, for example for different types of calorimeter cluster corrections, or track extrapolation. Abstract interfaces are used to reduce dependencies.

A rich set of algorithms is available. A number of algorithms were originally developed in FORTRAN for detector optimization, as documented in the ATLAS Physics TDR in 1999 [3-19]. They were then migrated to C++ and into the Athena framework, while new algorithms were developed. In most cases, two separate algorithms are available which allow for in-depth performance comparisons and cross-validation.

The event generator truth is used for optimizing and validating reconstruction algorithms: however, the policy is to do this in separate algorithms.

3.9.2.1 Tracking System Reconstruction

The tracking system reconstruction chain is summarized in Figure 3-7.

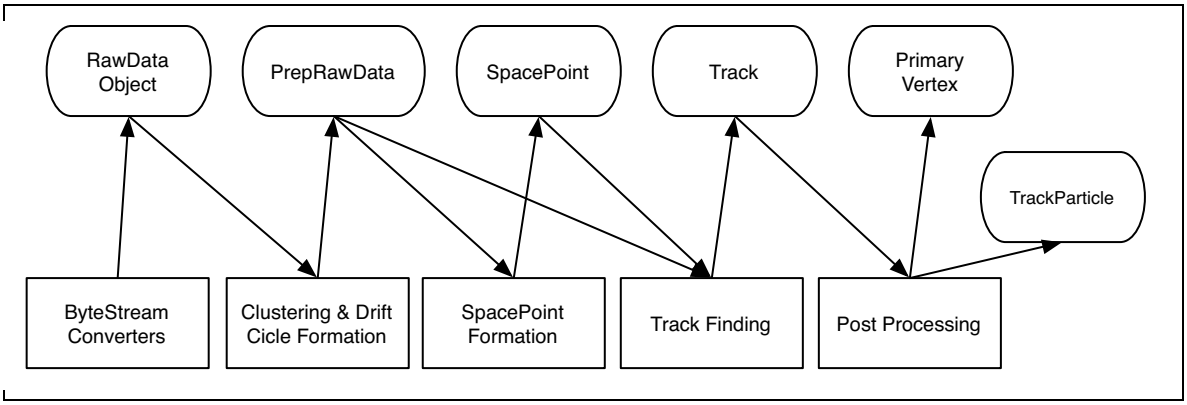

Figure 3-7 Tracking reconstruction chain. The boxes on the top represent data objects, whilst the boxes on the bottom show the algorithms which work on them. The arrows show the direction of data flow.

A basic requirement for the tracking EDM is to support different tracking devices with shared code, e.g. the muon chambers and drift tubes, the inner-detector transition radiation tubes and silicon detectors must all be provided for by common tracking software. The primary outcome of this requirement is a common track class, but, furthermore, the EDM needs standard definitions of:

- Track parameters (on all the various surfaces found along the track);
- Interfaces to hit-clusters, drift circles, etc.

Tracking must handle many different coordinate frames, as a track can span the entire detector and have measurements on many different surfaces (i.e. discs, planes, cylinders, and so on).

However, the various tracking tools and algorithms should not be expected to handle the geometry of the detector. Generalized tools allow tracking to work on both the Inner Detector and the Muon Spectrometer tracks. This can best be explained with the aid of the picture above, which shows an overview of the tracking reconstruction chain.

Byte-stream converters take the data from the detector, and form the raw data objects. These are then used to create "prepared raw data" (PrepRawData), i.e. clusters from the pixel detector or drift circles from the muon monitored drift tubes.

The PrepRawData (along with the SpacePoints) can then be used to find tracks. Finally, the tracks can be used to find vertices, and to create the TrackParticles (for physics analysis at the AOD level).

Clusters are searched for in the silicon tracker. Then tracks are searched for with two independent pattern-recognition algorithms, sharing a number of common tools. Silicon tracks are extrapolated and validated in the straw tracker. A dedicated algorithm examines the tracks found, and in case of duplication, keeps the one with the highest number of hits. A primary vertex is computed, and a set of track parameters extrapolated to the primary vertex is prepared. Muon track segments in the Muon System are found from a combinatorial search of the single-station track segments, followed by a fit using the single clusters. The tracking, performed in the highly inhomogeneous field, takes into account multiple scattering in the material of the apparatus.

3.9.2.2 Calorimeter Reconstruction

The two types of calorimeter have different data formats at the raw data level. However, for the reconstruction EDM, one common calibrated input object is used, CaloCell. CaloCells can be generated either from the raw data or simulation. For example, Figure 3-8, which is a schematic

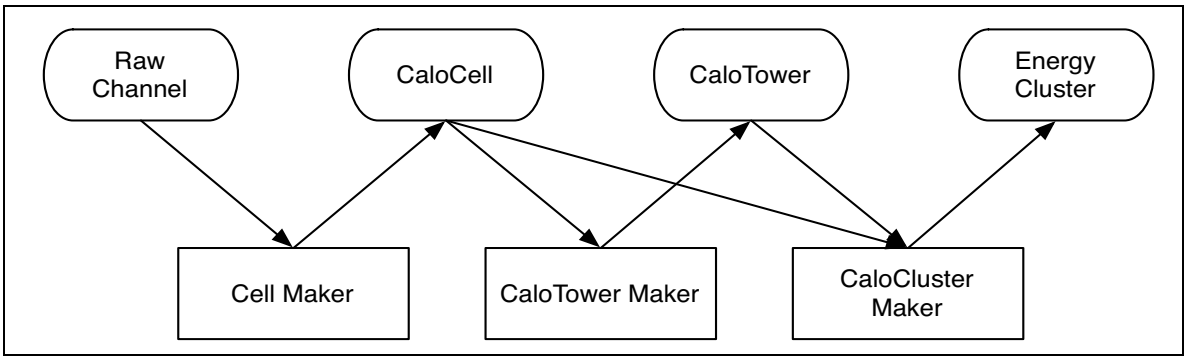

Figure 3-8 Schematic diagram of calorimeter reconstruction. The top line contains the data objects, whilst the bottom line shows the algorithms used to process them. Data flows from left to right.

representation of the calorimeter reconstruction chain, shows the raw data being fed to CellMaker algorithms, which produce CaloCells. After this reconstruction step the calorimeters use a common EDM. In particular, all calorimeter data classes inherit from a four-momentum interface; this allows the use of common tools that require only kinematic information.

Neighbouring CaloCells are used (by CaloTowerMaker) to produce calorimeter "towers", then these towers (as well as cells) are taken (by CaloClusterMaker) to construct "clusters", collections of calorimeter elements, which can even contain clusters themselves. A navigation scheme

allows access to constituent data objects; for example, it is possible to retrieve all the CaloCells used to create an EnergyCluster.

Cell energy is already available at the electromagnetic scale in the raw data. Refinement with calibration parameters is done as a first step of the calorimeter reconstruction.

Electromagnetic clusters can be reconstructed using different methods.

- The *sliding window* algorithm searches for the window where the total energy is maximum. The window can be adjusted to different sizes, so that it can be optimized for different particles/energies. They are then corrected for different modulation effects, and longitudinal weights are computed to further optimize resolution and linearity.
- The *topological clustering* algorithm attempts to aggregate neighbouring cells with signal above threshold over the complete calorimetry. The algorithm selects individual cells with an energy in excess of four sigma of the expected noise. Then neighbouring cells with energies in excess of two sigma are added iteratively. Finally a guard ring of cells without energy requirements is added. Proto-clusters obtained in this way are then split to separate local energy maxima. The intent is to explore the use of topological clustering, possibly with different tuning, for electron identification, jet reconstruction and missing E_t measurement.

3.9.2.3 Combined Reconstruction

The combined-reconstruction step combines information from the different detectors in an optimal way. The output EDM is designed to support a wealth of tagging variables from different algorithms.

3.9.2.3.1 Photon/Electron Identification

Electron reconstruction is performed in two ways. High-p_T electrons are searched for by associating tracks to sliding-window clusters, and computing shower-shape variables, track-to-cluster association variables, and TR hits variables. Dedicated track-fitting procedures for electrons are being developed. High-p_T photons are identified in a similar way, with the main difference being that a track veto is performed, except for reconstructed conversions.

Soft-electron reconstruction proceeds by extrapolating a charged track to the calorimeter, and building a cluster around the charged-track impact point. This procedure has a better efficiency for electrons with p_T less than 10 GeV, and for electrons inside jets, which is pertinent for b-tagging.

3.9.2.3.2 Muon Identification

Muon measurement and identification is optimized according to the p_T regimes.

High-p_T muons (>100 GeV) are measured by extrapolating the muon-spectrometer track parameters in the muon spectrometer inward through the calorimeters and inner tracker to the interaction point. Combination with the optimum inner-detector track may also be done. Methods to do this are being investigated. Such combination can be particularly effective where there exist acceptance gaps in the spectrometer, such as near $\eta = 1$. The extrapolation of the muon trajectory to the inner-tracker track allows computation of the energy loss through the in-

tervening material. Energy-loss parametrizations can be applied to correct the track momenta, as determined at the muon-spectrometer entrance, to the final-state muon momenta at the interaction point. Furthermore, direct measurement of catastrophic energy loss (important at high p_T) can be used to correct the muon momentum.

For muons in the 6–100 GeV p_T range, momentum determination is performed by both systems. The muon spectrometer provides a flag that uniquely identifies the muon. For momenta below 30 GeV, the measurement resolution derives mostly from the inner tracker as the muon-spectrometer resolution is dominated by multiple Coulomb scatters.

For p_T between 3 and 6 GeV, muons lose a large fraction or most of their energy in the calorimeters, and do not cross the full muon spectrometer and therefore cannot be reconstructed there. In this case, muon tracks are found in the inner detector and extrapolated to hit segments in the spectrometer. Algorithms that extrapolate inner tracks and associate them with a minimal signal in the Inner muon station (e.g. a solitary segment in a multi-layer) are being developed. Muon identification at low p_T can also be enhanced via signatures in the tile calorimeter. This is being investigated.

3.9.2.3.3 Tau Identification

Taus are identified in a similar way to electrons. The preliminary clustering is done with a sliding-window algorithm applied on all calorimeters. A tau appears as a very narrow jet in the calorimeter, associated to a small number of charged tracks.

The tau reconstruction can be seeded by a calorimeter cluster or by a charged track depending on the p_T range of interest.

Tau identification is based on calorimeter quantities such as the electromagnetic radius, the isolation in calorimeters, the width in the strips and on quantities given by the tracker such as the number of associated tracks, the charge and the impact parameter. Likelihood and multi-variate analysis techniques are used to discriminate taus from normal jets.

Taus are calibrated using the same cell-weighting scheme as jets.

3.9.2.3.4 Jet Reconstruction

Jets can be reconstructed from detector signals, and for Monte Carlo data, from the generated particles. The algorithms available are the seeded and the seedless cone and the k_t algorithm. The cone algorithms have native implementations in ATLAS software, following the guidelines given in Ref. [3-30]. The k_t implementation is provided in an external package [3-31], which is wrapped by a specific tool creating the Jet objects of the ATLAS EDM. In the implementation there is only one jet algorithm skeleton, which can be configured externally as a sequence of tools to implement a given jet-finder strategy. This algorithm and most of the tools are designed such that they are not dependent on any specific feature of the input data objects, thus allowing their use in exactly the same way for different inputs. The only requirements on the input objects are that they implement the general four-vector and navigation interfaces.

3.9.2.3.5 Calorimeter Jets

The calorimeter system is the principal detector for jet reconstruction. The typically large number of CaloCell objects in an event prohibits using these directly as input to the jet finding, especially in the case of the k_t algorithm. The input multiplicity to the jet finding can be reduced by the calorimeter reconstruction, where cells are grouped into CaloTower and CaloCluster objects. The CaloTower objects represent a tower of cells on a fixed grid in pseudo-rapidity and azimuth, typically with a bin size of $\Delta\eta \times \Delta\varphi = 0.1 \times 0.1$ for input to the jet finding. CaloCluster objects, on the other hand, represent groups of cells with correlated signals with their location depending only on the cell signals and locations. Both CaloTower and CaloCluster implement the four-momentum and navigation interfaces, as required by the jet algorithms.

All jet algorithms combine the input object into a Jet object following their specific strategies. The total jet kinematics is represented by a four-vector, which is updated when constituents are added or removed. This four-momentum recombination requires all constituents to have meaningful four-vectors themselves, especially a positive signal amplitude (energy). On the other hand, CaloTower objects can have negative signals, indicating a major noise contribution from the cells in this tower. These negative-signal towers are combined with neighbouring towers until the newly created combined tower has a small positive signal, thus cancelling the negative signals before applying the actual jet finder.

Calorimeter jets can be calibrated in various ways. The standard calibration for jets from towers is based on a cell-signal weighting scheme, where weights are applied to the signal contribution from each cell. These weights have been computed such that the response to jets is flat over a large energy range, and using the constraint of an optimized energy resolution. Other approaches apply weights to calorimeter-sampling layer sums in jets, for example.

Truth Jets

Jet finding using the generated particles requires the retrieval of these particles from the truth event associated with the reconstructed event in simulations. The jet reconstruction provides a special tool for this task. After the extraction, all tools used for jet finding in the detector can be used in exactly the same way. In particular, all pre-clustering and pre-sorting, as required by the k_t algorithm, for example, is done using identical software.

3.9.2.3.6 Missing E_t Reconstruction

Missing E_t is reconstructed from the energy deposed in all calorimeter cells and from the reconstructed muons. A correction is applied for the energy lost in the cryostat between the electromagnetic and hadronic calorimeters.

The calorimeter cell energy is weighted using the same H1-style weights, depending on cell energy density (E/V) and on the calorimeter region, used for jets. For muons the reconstructed energy from the muon chambers only is used, to avoid double energy-counting in the calorimeters. The correction for the energy lost in the cryostat is calculated from the energy deposited in the cryostat by jets.

To suppress the effect of noise in calorimeters, a cell energy threshold in terms of number of sigma noise is applied.

Missing E_t can alternatively be reconstructed from the energy measured in the topologically clustered calorimeter cells. In this case the noise suppression is given by the thresholds applied in the topological clustering reconstruction.

3.9.2.4 Reconstruction Performance and Prospects

The ATLAS Computing Model described in Chapter 2 assumes that reconstruction (creation of the ESD) requires approximately 15 kSI2k-sec per event processing time. In the absence of pile-up, the current measurement is approx. 22 kSI2k-sec. However, not all of the algorithms have been optimized, and in several cases multiple versions of some algorithms are run (e.g. tracking, calorimetry cluster finding, jet reconstruction), so it is expected that the design goal can be achieved. The situation in the presence of pile-up is worse, but again is expected to be amenable to significant optimization prior to deployment.

Continued development of reconstruction algorithms prior to data taking focuses on the following aspects:

- optimize performance;
- render algorithms robust with respect to the real data-taking conditions: varying calibration and alignments, noisy/dead channels, etc. With this goal, reconstruction algorithms have been adapted to run on combined test-beam data taken in 2004 (analysis is under way). Commissioning data will also be used as soon as it is available (autumn 2005). Also the simulation is made more and more realistic by allowing deterioration as expected in the data;
- adapt algorithms to work in the HLT context;
- continue validation and tuning of existing algorithms for varying conditions (in particular low-luminosity pile-up) and for different analysis;
- development of new algorithms to extract the maximum amount of information from the data; for example low-p_T particle identification, multiple interaction in the tracker.

3.9.3 Analysis Preparation

3.9.3.1 AOD Building

The analysis preparation consists of the production of AOD and the collection of interesting events for analysis.

The Event Summary Data (ESD) contains the persistifiable output of the combined reconstruction described above. The Analysis Object Data (AOD) is produced from the ESD by using very loose selection criteria applied on objects such as the reconstructed photons or electrons. In addition new AOD-specific objects, e.g. the JetTag, are created. The objects in the AOD can be redundant, can overlap and can be ambiguous. Some of these overlaps are removed during the AOD making, but not all, as the details of the removal may be physics-analysis-dependent.

The event collections are pointers to events in persistent storage, along with event-level metadata (the tags) used for event selection. The collections are arbitrary: they may span many streams, and a given event may appear in different collections. Physics groups and users may extract copies of their interesting events by querying the tag database and filling their dedicated AOD

samples. They can also build new collections (probably ROOT-based) containing only the results of the selection.

For fast simulation, the tools exist to produce the AOD using as input generated events, simulated events, digitized events or the ESD, since each of these data formats contains the full record of the event generation, used as input to the standard fast simulation.

Figure 3-9 shows a generic analysis object. Since it represents a physical object, it inherits from a four-momentum, navigation and a basic particle class, where the navigation interface allows (in the same manner as the calorimeter objects) navigation between constituent objects. Pointers to ESD objects are saved, which allow direct navigation to detailed information in ESD if deemed necessary.

Tools which require only kinematic information will just use the four-momentum interface, whilst other analyses might need more detailed information. In any case, the use of common interfaces dramatically simplifies the analysis code.

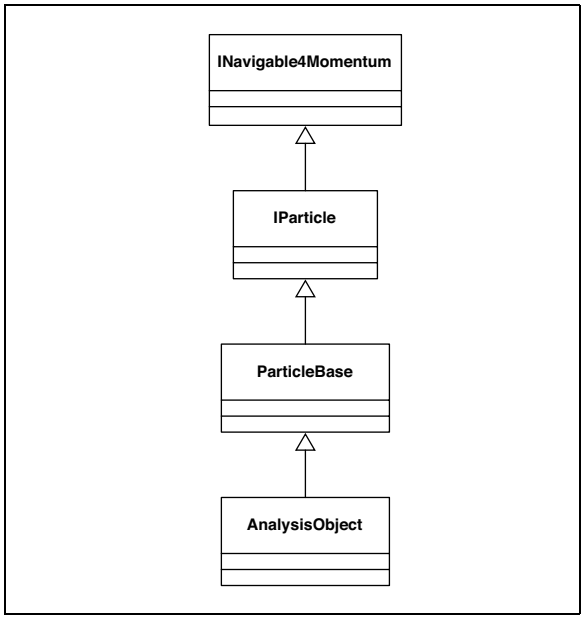

Figure 3-9 An analysis object and its inheritance structure. Examples of such objects are muons, b-jets, taus, etc.

3.9.3.1.1 Electrons, Photons and Muons in AOD

The electrons and photons selected for the AOD are required to have a hadronic-energy fraction of less than 20%. Electrons and photons in the AOD can never overlap because of the requirement of a matching track for the electron, and of no matching track for the photon. The overlaps between the collections of high-p_T and low-p_T combined-reconstruction electrons and muons are also removed when making the AOD; when a low-p_T candidate shares a track with a high-p_T one, the high-p_T candidate is kept. The development of the tools needed to deal with the redundancies, overlaps, and ambiguities in the AOD is discussed in Section 3.10.

3.9.3.1.2 Taus in AOD

The tauObjects reconstructed by the tauRec package and found in the ESD are converted 1-to-1 into the AOD TauJet container. The TauJet object keeps only the most important variables for refining pre-selection (likelihood, number of tracks, hadronic, and electromagnetic energy etc.).

3.9.3.1.3 Jets in AOD

Jet objects within the ESD are converted 1-to-1 into the AOD ParticleJets. This is done for cone jets of R = 0.7 and R = 0.4 as well as k_t jets. In addition, some calorimeter information is computed from the ESD jets and stored in the AOD. The ESD jets are converted into the JetTag objects described in the next section. In the future, the two classes will merge to form a single jet class for the AOD.

3.9.3.1.4 b-Tagging

Identification of jets containing decay products from bottom-flavoured hadrons, or *b-tagging*, requires jets with tracks. This implies that the calorimeter jets cannot be used directly. A new jet-object has to be constructed using track objects as well as calorimetric information. The b-tagging output object, the JetTag, is stored in the AOD. Users have the possibility to re-run the tagging on AOD without having to navigate back to information stored in the ESD files.

The relatively long lifetimes of b-hadrons can give rise to displaced vertices. The “secondary” vertices can be tagged by examining the impact parameters of the tracks in the jet. B-jets have a characteristic long, positive tail in the distribution of impact parameters; for the “light” jets (from the light quarks) one expects a symmetrical distribution. Another method is to explicitly reconstruct the secondary vertex using vertex-finding algorithms. If there are two or more tracks in the jet with a significant impact parameter, a secondary vertex can be searched for exclusively. Properties of the secondary vertex, for example, the fraction of the jet energy and the reconstructed vertex mass, can be used to discriminate b-jets from “light” jets. In addition, “soft” leptons (from semi-leptonic decays of B's) can provide a limited but valuable complement to the space tagging (see above for “soft” electron and “soft” muon identification). The results of all methods can be combined into one single discriminating variable in different ways, using different test-statistics and combination methods.

3.9.3.2 Size of the ESD and AOD files

One critical issue is the actual file size of the EDM objects written to disk. The goals for the size of ESD (500 kB/event), AOD (100 kB/event) and data tags (1 kB/event) have been established based on experience from earlier n-tuple-based reconstruction data, and the consideration of the cost of storage. The size and content of the ESD/AOD is currently evolving quite rapidly due to the increasing knowledge of what is actually needed for analysis and the increasingly efficient storage of information. In addition, the size is heavily dependent on the physics process and LHC luminosity. Finally, events produced from simulated datasets have an additional component, the Monte Carlo truth, which allows a detailed comparison to be made between the results of reconstruction and the original event. This additional information significantly increases the event size, but is crucial in arriving at a detailed understanding of the performance of the reconstruction software.

At the time of writing, the 500 kB target size for the ESD has not been reached, with the current size being approximately 1.2 MB/event. However, the number of objects in the ESD has intentionally been chosen to be as inclusive as possible so as to not inhibit any physics analysis. Feedback from the physics community based on operational experience and further work on storage techniques are expected to allow the size to be reduced to meet the target before data taking starts.

The detailed content of AOD and ESD is shown in Figure 3-10.

3.10 Physics Analysis Tools

The Reconstruction Task Force (RTF) examined the architectural aspects of the ATLAS reconstruction software, and arrived at a set of suggestions/proposals which have come to be known as the RTF recommendations [3-32]. The RTF covered the preparation of data for physics analy-

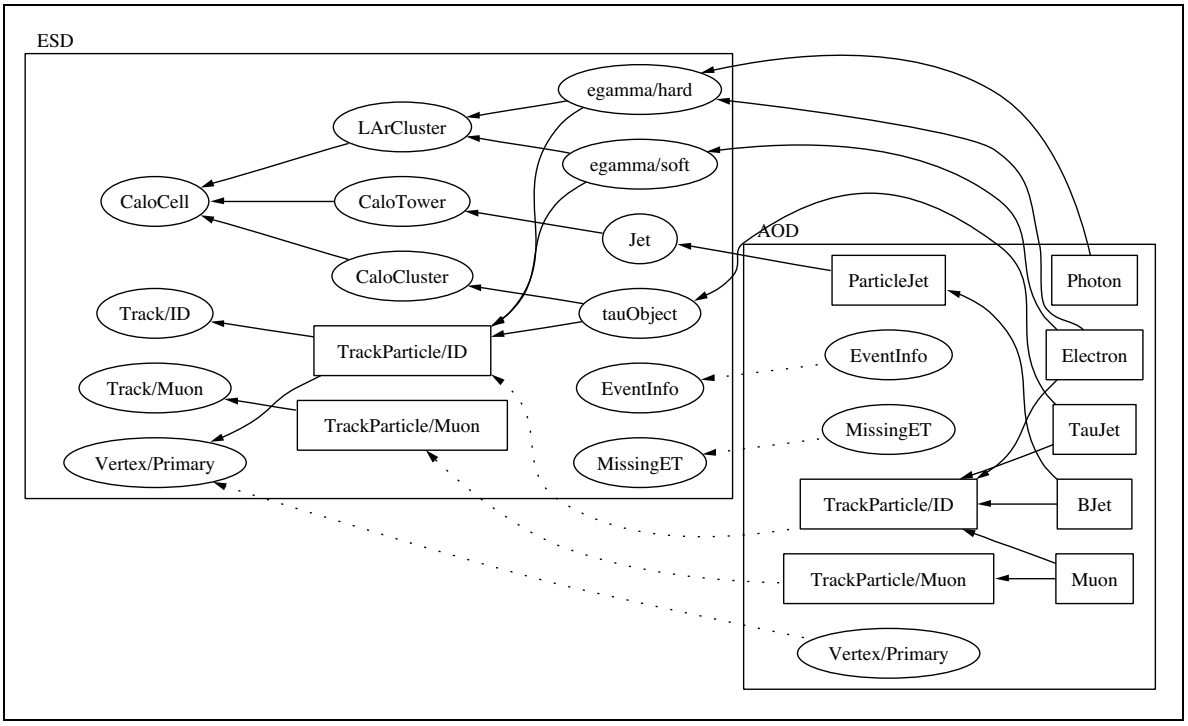

Figure 3-10 Simplified ESD and AOD content. The solid lines indicate direct navigation possibilities. The dotted lines indicate duplication of objects.

sis, but its mandate did not extend into physics analysis in detail. The scope of Physics Analysis Tools (PAT) is to span the gap between the combined reconstruction and the analysis on n-tuples. The objectives of PAT include:

- Development, within the Athena framework, of an environment where reconstruction tools are available, while at the same time taking advantage of n-tuple analysis tools such ROOT [3-7], PAW [3-33], JAS [3-34].
- Support for both batch and interactive analyses.

The activities carried out within the PAT group are not about ROOT, PAW, JAS, etc. as standalone analysis tools, nor about distributed physics analysis. Rather, the idea is to propose a unified, baseline framework for analysis, to explore and propose various options for interactive analysis, and to interact with combined performance and physics groups so as to develop tools satisfying user requirements. The unified, baseline framework for analysis consists of the following:

- Common classes in the analysis domain, such as the AOD (Analysis Object Data).
- Common tools to build these objects, such the AOD builder algorithms.
- General and common tools for analysis.
- Navigation and association tools.
- Tools for overlap checking, redundancy removal, and ambiguity resolution.
- Tools for event views.
- AOD streaming, event tag and event collection tools.
- Tools for interactive analysis and event display

- Documentation.

A dedicated workshop was held in April 2004 to make a first attempt at a baseline, unified, common framework for analysis, thus extending the work of the RTF. A detailed summary of the workshop can be found at reference [3-35]. A follow-up workshop was held in May 2005.

3.10.1 Current Status

Baseline implementations of all the AOD classes exist. The contents of the ESD, AOD and event tags follow the recommendations of the AOD/ESD definition task force [3-36] and of the combined performance groups. The PAT group provides a set of tools useful in the analysis environment. They consist of the following tools:

- Combination, permutation with or without selection criteria – for example, making jet–jet combinations selecting only the combinations that pass pre-defined criteria.
- Sorting – to sort any user collection of objects.
- Filtering according to different criteria – for example filtering the MC event collection to search for a particular decay pattern.
- Constituent navigation. The original motivation for object navigation came out of jet reconstruction. Jet constituents are of generic type, their concrete type is not exposed to the jet itself; clients need to retrieve objects of specific concrete type at any node of the relational tree behind a jet, thus a navigation system is needed. Constituent objects in the tree can be composites themselves and thus navigation must be possible to any given level in the tree. Constituent objects can contribute their kinematics with a weight to the composite object and thus the weights must be retrievable and propagated correctly.
- Back navigation. Not all the objects that the user might need at the analysis stage are available in the AOD. When the requested object is not found in the AOD, the process which searches for the object in ESD or even in the raw data is known as back navigation.
- Composite particles. For example, the Z-boson as a composite of an electron–positron pair with all the constituent navigation features from the Z-boson to the calorimeter clusters or cells of the electrons.
- Association tools, non-constituent associations. For example, a muon can be associated with a jet without belonging to the jet. One may wish to associate a muon to a jet for the purpose of b-tagging but also associate a further muon to the same jet as a candidate for the decay of a top quark.
- The user analysis package. As a part of the analysis, to help the user get started quickly with his analysis code, a user analysis package is provided in the CVS repository under PhysicsAnalysis/AnalysisCommon/UserAnalysis/. It contains a skeleton analysis algorithm and sets up the CMT environment for the user. The idea is to provide an environment where novice users can get started quickly in developing their own analysis code.
- Interactive analysis in Athena. The PAT group also provides tools for interactive analysis. It is possible to browse the content of the AOD and make plots of the raw AOD data. It is also possible to examine the raw AOD data interactively without writing a single piece of code. The user may define, fill and manipulate histograms and n-tuples, and it is possible to access the histograms and n-tuples that are defined in the analysis algorithms.

- Analysis in Python. The tools exist for the user to write complete analysis codes in Python.
- In collaboration with the database group, the PAT group provides tools for the event tag definition, the AOD streaming, and for collections of interesting physics events.
- Special utilities. The PAT group provides tools to address specific class of problems. For example, to solve for neutrino objects in $X \rightarrow \tau\tau$ or in $W \rightarrow l\nu$ using the collinear approximation or the W-mass constraint, respectively.

3.10.2 Short-Term Objectives

Some of the tools described above are being improved with added functionality, following requests from individual users or groups. Concurrently, the design and implementation of other tools are being discussed. These consist of the following:

- Overlap/redundancy/ambiguity. A set of tools to check for overlaps between different objects in the analysis domain; to remove redundancies (for example, the collection of jets includes the jets that are already tagged as b-jets and put in separated b-jet collections); to solve ambiguities (an object reconstructed as both a high-p_T and a low-p_T muon, which thus appears in both collections).
- SymLink is a tool which allows one to record a container of objects as one type and retrieve it as a container of a different type. For example, through SymLink, one can record a container of electrons and later, for whatever reason or purpose, retrieve the same container as a container of IParticles – the Electron and the Muon classes derive from the IParticle class but the Electron container or the Muon container classes do not derive from the IParticle container class. How to relate the Electron container to the IParticle container is known as SymLink. Many analysis use-cases of SymLink exist: for example, when one would like to treat the container of Electrons and Muons as containers of IParticles with no regard to the detailed differences in the Electron and the Muon implementations. Although the current implementation of the SymLink works for most use-cases, it does not work in certain circumstances and it is not compiler safe nor portable.
- A fast simulation tool, known as the Atlfast comparator, is being developed. The objective is to tune the fast simulation parameters by making detailed comparison with fully simulated or real data. This is described in Section 3.8.3.3 above.
- One needed feature of the interactive analysis is the ability to find and read-in an arbitrary event from the input data stream; in general, the ability to re-initialize the event loop without exiting the interactive session. A prototype tool to do this exists, PyPoolSeek. The extension of this tool for the run number in addition to the event number, i.e., seek(run number, event number), and for suppressing the processing of intermediate events when one needs to get to a specific event would be good to have and should be investigated. Two other features are needed to make the interactive analysis truly useful. One is adequate processing speed, i.e., to be able to run over a moderately-sized sample and make plots in a few seconds. At the moment, we are at the level of a few minutes; a caching schema should be considered to improve the processing time. The other needed feature is the ability to read different multiple samples in a single interactive session, e.g., open both signal and background samples, make histograms on each and overlay the histograms.

- Interactive analysis and event display: Atlantis[1] is moving in that direction so that it will be able to do everything available in interactive Athena. The development surrounds three related technologies: interactive analysis in Athena, the XML RPC Server, and Atlantis. One can run a remote interactive Athena session and steer it with XML RPC. In that set-up, one asks the server to execute an interactive Athena command on a remote interactive Athena session. Essentially, this supplies the reverse form of the communication. The plan for Atlantis is to be able to transmit information back to the interactive prompt.
- Event view. This is a coherent and exhaustive list of physics objects that are mutually exclusive. The user may wish to consider different views of the same event. By coherent, it is meant that the user does not need to carry out additional checks nor call additional tools to guarantee the self-consistency of the view. The sum of the energies of the objects in the view should come out as the total energy in the event; the total sum of the transverse momenta, including the missing transverse momentum, should go to zero: the view is exhaustive when these criteria are met. Objects in the view are mutually exclusive; e.g. a jet should not also be listed as an electron. Thus, the overlap checking, redundancy removal and ambiguity resolving tools are integral parts of the tools for the event views.

The analysis tools [3-37] described here are already quite well documented. Web and wiki documentation is available on the PAT web page [3-38]. This page is linked from the ATLAS main page and also from the ATLAS computing page.

3.11 Use of Offline Software for HLT and Monitoring

The ATLAS High-Level Trigger (HLT) comprises the Level-2 trigger (LVL2) and the Event Filter (EF) which provide the second and third levels of the online event selection. These run in farms of Linux PCs (the first-level trigger, LVL1, using coarse-grain data from the calorimeters and muon detectors, is implemented in custom electronics). The second-level trigger runs specialized algorithms using selected fine-grain data to confirm and refine the LVL1 trigger, and the Event Filter runs optimized offline algorithms to provide additional confirmation and refinement. Overall the HLT has to provide the required rate reduction from the 75 kHz of LVL1 to the 200 Hz which can be recorded for offline analysis.

Key features of the HLT are:

- Specialized HLT applications provide the framework which interfaces to the online run control and provides the data movement.
- All of the online state knowledge is contained in these applications.
- The selection algorithms see an Athena-like environment running inside these applications – this allows various offline software components to be used in the HLT and simplifies offline testing and development of the selection code.
- Data from previous steps are used to seed the algorithms.
- Algorithm data are unpacked only when needed.
- All initialization required for the algorithms (calibrations, alignment, etc.) is performed at configuration time at the start of a run.

1. Atlantis is described in more detail in Section 3.6.2

As noted above, the HLT applications provide the interfaces to the online run control and provide the data movement, with different implementations being used for the LVL2 and EF systems. In the LVL2 trigger, data is received as event fragments which are requested across the network from the data-acquisition ReadOut Buffers as required. Typically only a few percent of the data in an event is required, thus saving both data-bandwidth and processing time. In contrast the Event Filter DataFlow provides complete events in byte-stream format.

The use of the Athena-like environment in both LVL2 and EF allows considerable commonality in the event selection software itself and also allows this software to use various offline software components, provided that other requirements of LVL2 and EF are also met. Because of the short processing-time-budgets and the larger numbers of events processed, both areas impose severe performance and robustness requirements, particularly so at LVL2, where there is also a requirement for thread-safety as the LVL2 code runs in a multi-threaded environment.

Offline components currently being used within the HLT include:

- The detector description;
- Calibration and alignment;
- The event data model and byte-stream converters.

In some cases modifications are required, for example to meet the performance requirements, especially in LVL2. Figure 3-11 shows the package diagram for the Event Selection software. This use of offline components within HLT leads to dependencies between the HLT and offline software repositories and places additional requirements on those offline components used. The mechanisms used to handle the additional dependencies are described in Section 3.14.10. In addition specific HLT tests are included in RTT (described in Section 3.13) to test new offline releases.

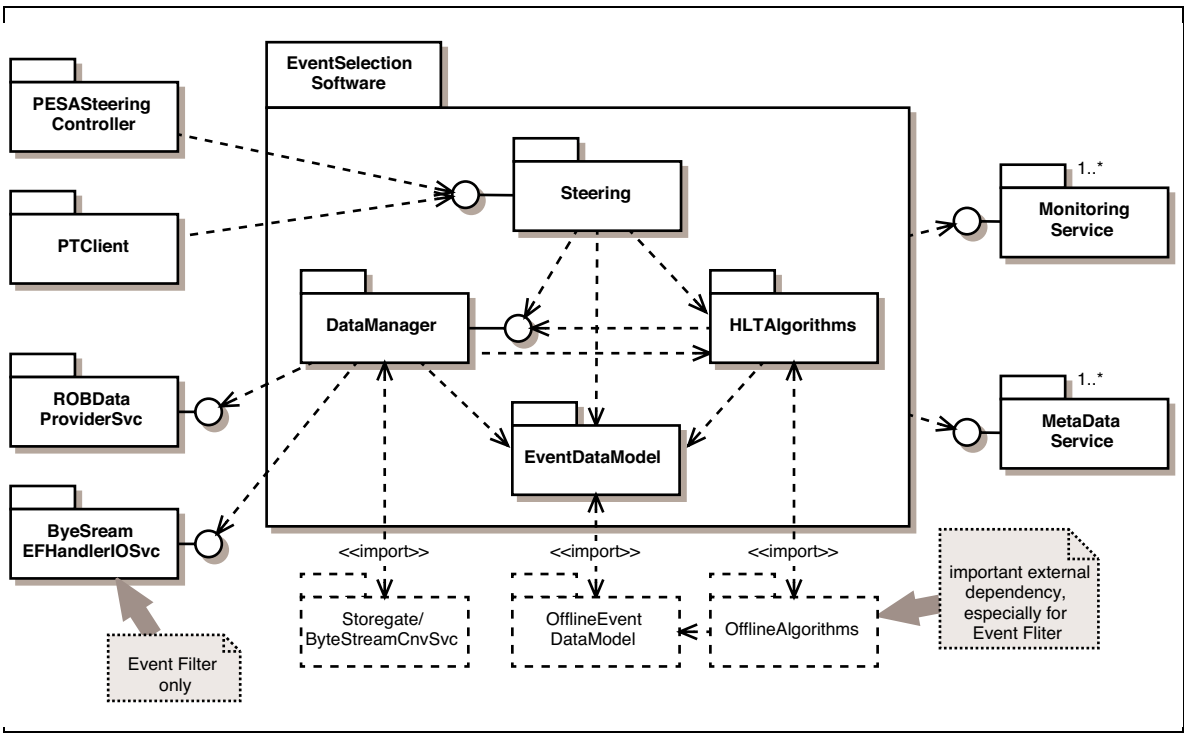

Figure 3-11 Package diagram for the Event Selection Software.

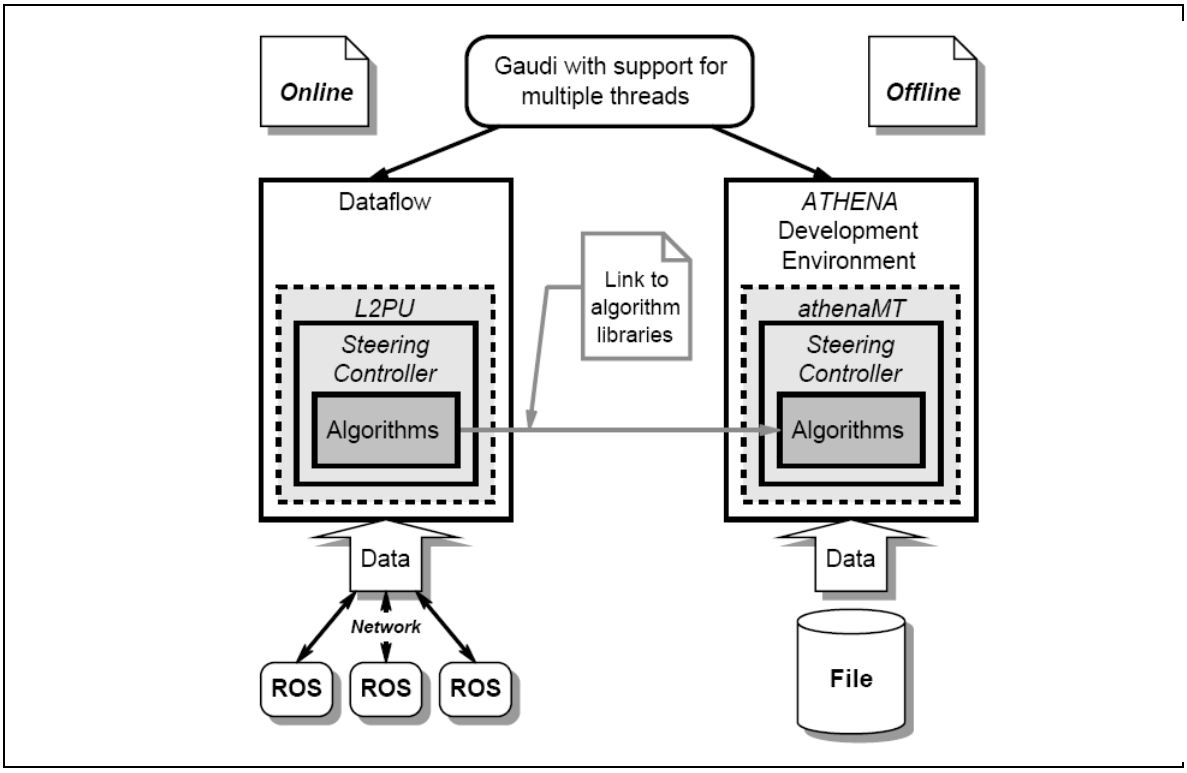

Figure 3-12 Online and Offline components.

To test the offline components for compatibility with the Athena environments provided at LVL2 and EF, the additional offline tools AthenaMT and AthenaPT have been provided – AthenaMT includes tests of thread-safety. This is shown in Figure 3-12.

The EF provides an infrastructure whereby complete events are provided to a processing task where selection algorithms see an Athena environment. This same infrastructure can be used for data monitoring by simply replacing or augmenting the selection algorithms by those for monitoring. This has been further generalized by allowing the data-input to be received not

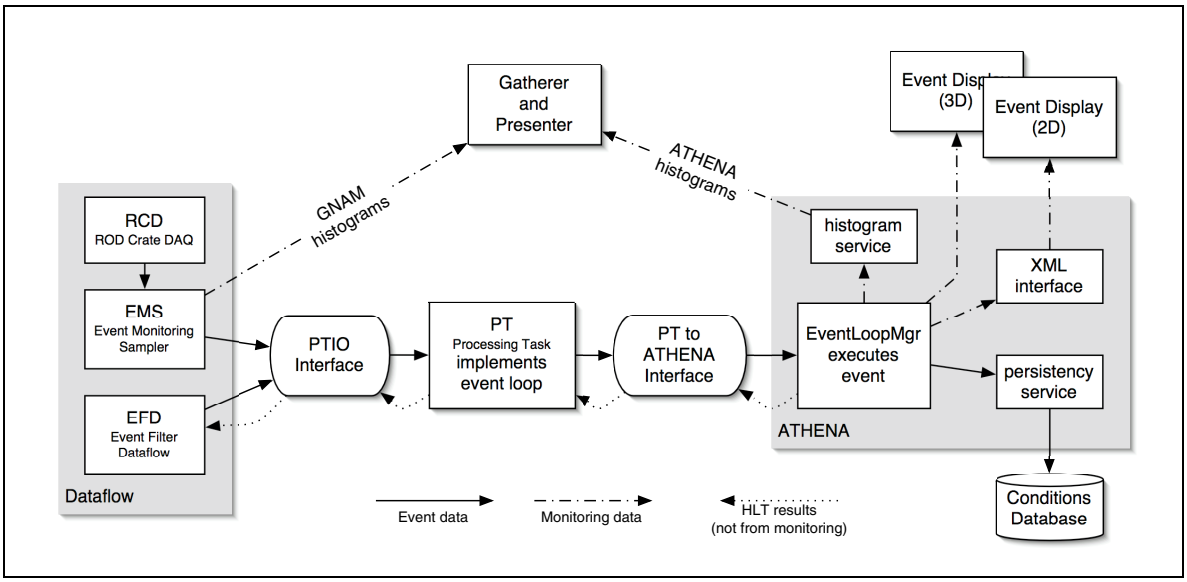

Figure 3-13 Integration of Athena for online monitoring.

only from the Event Filter DataFlow, but also from the Event Monitoring Sampler – as shown in Figure 3-13. This capability will be used extensively during Detector Commissioning as described in Section 3.12.

3.12 Use of Offline Software for Detector Commissioning

Commissioning the ATLAS detector will be a major focus of the Collaboration for the next few years. The use of offline tools, including the offline software, computing and databases, can both facilitate the detector commissioning and ensure that the offline is itself ready for ATLAS data analysis from the earliest collisions.

The approximate time-scales of detector commissioning activities are outlined in Table 3-1.

Table 3-1 Time-scales of ATLAS detector commissioning activities.

Phase	Activity	Approximate dates	Needed tools
1	Front-end electronics installation	Spring 2005 to Summer 2006	Byte-stream converters, histogramming service, conditions DB, file management, event displays
2	Sub-detector integration	Autumn 2005 to Autumn 2006	Integrated commissioning releases
3	Cosmic runs, single beam	Autumn 2005 to Spring 2007	Reconstruction of out-of-time, non-pointing events

A critical first element for the use of the offline software for detector commissioning is the access of data in Athena, both event data and conditions data. Detector data flow paths are sketched in Figure 3-14.

Athena can access event data online on the High-Level Trigger processors, be run on raw (byte-stream) data files, or run on any system accessing events via the TDAQ “event sampler”.

Initial ATLAS “phase 1” commissioning activities will proceed within each subsystem independently. At this stage detectors will be installing front-end electronics, and taking local pulser or calibration triggers to verify successful installation. The data, primarily written using the VME readout and the ROD crate DAQ (RCD) system, will be written in standard ATLAS byte-stream files that can be analysed from Athena or other software. At this stage, it is critical that the system byte-stream converters are able to decode the detector fragments from the byte-stream, and that the software environment be flexible enough to keep pace with potentially rapid changes in the detailed event format as the first data is recorded. The first use of these data will be to verify the functionality of the front-end and readout electronics. Relatively little reconstruction will be required initially, but a detailed data-monitoring, event-display and histogramming environment will be important. The monitoring tools and “AthenaMonitoring” environment developed for the 2004 combined test-beam tests (CTB) can potentially be extended for use for detector phase-1 commissioning activities. It is also important that detector byte-stream converters and monitoring tools be archived and maintained in the central ATLAS CVS repository, ensuring their availability for future use.

Event-data files will already require file management and archiving from the early commissioning phases. These files will typically be written from an online workstation to a local disk, and will be analysed both locally and remotely. Aggressive migration towards prototypes of the ATLAS data-management system from the early commissioning stages will ease file access with the offline software and also provide file and run book-keeping tools.

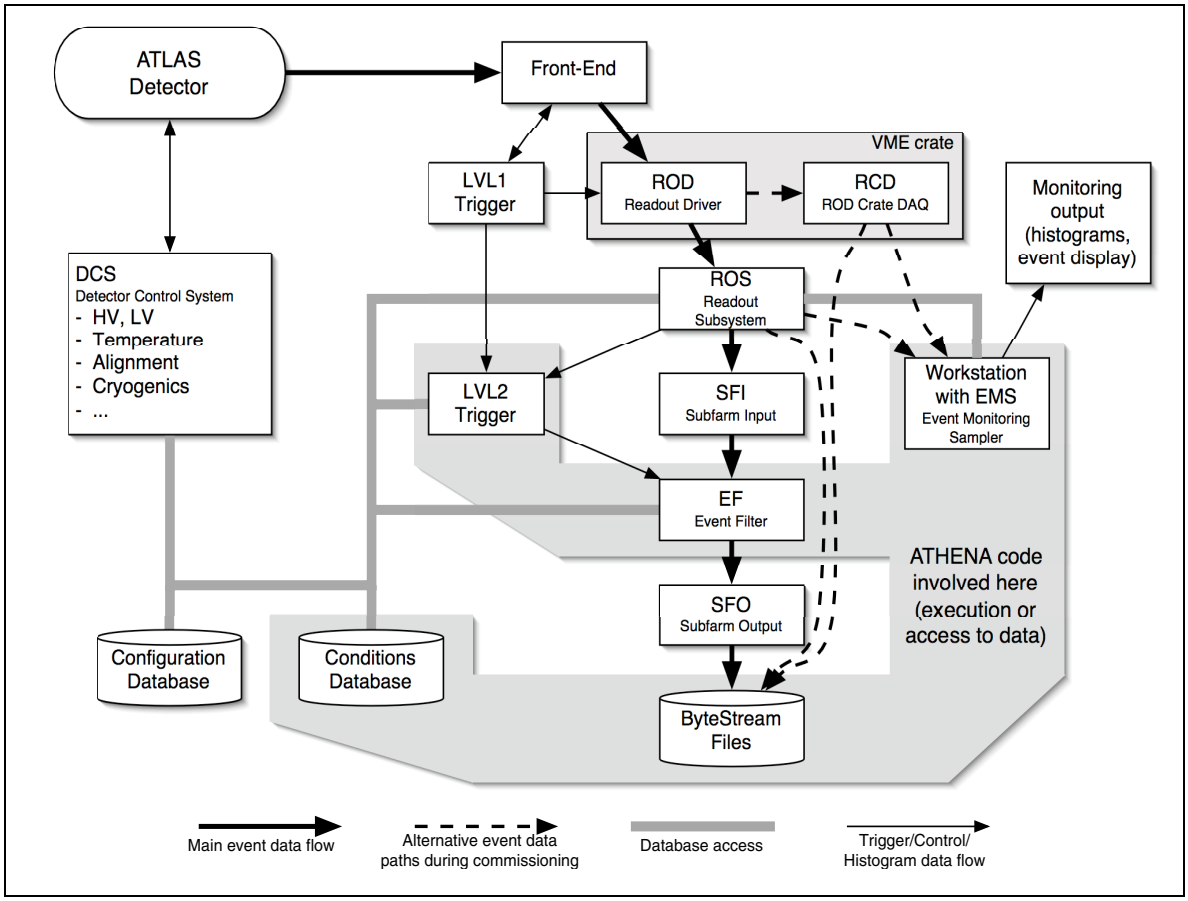

Figure 3-14 Data flow of events and conditions data.

The initial data runs will produce calibration data, which in some cases can be used for the production of calibration constants written to the conditions DB. This requires the extension of detector-calibration software to the full set of channels present in ATLAS, and also the use of final ATLAS conditions DB tools such as the LCG COOL conditions database and POOL file persistency.

In addition to event-data access via files, the offline software can also be run on the high-level trigger or other online workstations. Any online machine on the ATLAS control room network (ATCN) can access event fragments via the TDAQ online "EventMonitoringSampler" (EMS). The EMS was used successfully for certain monitoring applications in the 2004 test beam, although not using Athena-based monitoring applications. Recent developments have made it possible to access data via the EMS from Athena, as illustrated in Figure 3-13. Via the EMS, it is possible to analyse data using the same offline tools from any stage of the TDAQ readout chain. These monitoring tools can be used from the earliest commissioning phases.

After (and even during) phase-1 commissioning, the different subsystems will be integrated together into combined events, called commissioning phase-2. At this stage, it is obviously critical that all system byte-stream converters and monitoring tools run in a common software release, and the maintenance of this commissioning release will be important. It is also important that all systems are using common database tools by this stage. Dedicated updates of the ATLAS releases and use of CVS during phase 1 will make this transition smoother.

Commissioning phase-3 starts with cosmic-ray muon runs, and continues with single-beam running where beam-halo muon and beam-gas events can be collected. The rates of these events have been studied [3-39], and appear to be very useful for detector commissioning and early calibrations. These events are useful for channel-timing adjustment, detector alignment, and energy calibration. First cosmic-ray data taking will start in the autumn of 2005 with sector 13 muon-chamber cosmic runs in the ATLAS pit and SCT+TRT barrel runs on the surface. The tile calorimeter barrel will be fully instrumented and able to participate in common cosmic runs with the muon system in late 2005, and the liquid argon barrel calorimeter will be able to join in early 2006. The endcap calorimeters, inner detector barrel and endcaps, and the pixel system will be installed and ready throughout 2006 and early 2007, and a full combined ATLAS cosmic run will be possible in spring 2007. Single-beam running is scheduled to take place by the summer of 2007.

The reconstruction of cosmic-ray, and later beam-halo, muon events requires special tools in the offline software. These events do not arrive synchronously with the LHC (25 ns) clock, and are not pointing to the detector origin. They are, in some aspects, more analogous with test-beam data than LHC collision data. Tools will be required to reconstruct the event time for use in system reconstruction; tracking software must not require pointing tracks and must accommodate the different "sign" of energy loss, and calorimeter reconstruction must cope with the asynchronous events. While significant work has been done in these areas using either the commissioning MC samples described in Ref. [3-39] or using test-beam data, more effort is required to allow the full utilization of the commissioning samples.

3.13 Testing and Validation

3.13.1 Testing Aims

The aims of the testing infrastructure are:

- To provide tools that perform common tasks, allowing developers to concentrate on providing those aspects of testing that are package-dependent.
- To provide a standard way of specifying tests and handling test results.
- To identify testing roles (both human and computer), and ensure provision of all the components needed for testing to be carried out.

3.13.2 Types of Testing

ATLAS offline testing activities to date include unit testing, small-scale integration testing, large-scale integration testing, and physics validation.

Both the level and type of testing has been left up to the developers. The testing group has concentrated on building test frameworks to ease the burden of testing, thereby encouraging developers to provide tests, and allowing management to monitor the level of testing and the test results themselves. The various types of testing are:

- Unit tests, or simple and fine-grained tests that typically deal with individual classes or functions. The CppUnit tool is provided for automation of C++ code unit tests. Providing

complete unit tests for a package involves a large amount of work, but provides high confidence in code quality.

- Small-scale integration tests do not strive for complete code coverage. Rather, they test some small amount of functionality. The implicit assumption of such tests is that success infers that an unspecified amount of code needed to generate the result also ran successfully.
- Large-scale integration test jobs, which run on computing farms, usually perform regression tests on the output of a physics job such as text files, histograms and n-tuples.

The test frameworks have also been a useful way of regularly running standard jobs where the testing step is replaced by a human examination of job output, usually log files. These frameworks are also used to run quality-assurance jobs which run source-code metrics and check adherence to coding standards.

3.13.3 Testing Frameworks

As a result of the need for offline testing, work started independently on three separate frameworks. These are AtNight (ATN), Kit Validation (KV) and the Run Time Tester (RTT).

While each framework runs tests and displays their results, there are differences in the intention as to how these frameworks are to be used. Often the tests use the results of a job – such as an Athena job – as input. This requires the job to be run before the tests can be carried out.

All frameworks can be run locally, where the developer has complete control of the framework. Local running allows the developer to run tests on code that has been checked out of the repository and modified.

ATN and the RTT are also run regularly on the nightly and numbered releases. Both run KV as part of this procedure.

In practice, the test frameworks are run in different environments. This is not a fundamental necessity, but is a result of the differing emphases of the framework developers. ATN is run on the machines used to run the nightly builds. This limits the resources available for testing, and tests run under ATN are usually short. KV testing is usually run “by hand” as part of kit building, and once again the tests are usually short. The RTT is currently run daily on a farm at UCL. With these resources, more extensive testing is possible. The full set of RTT results currently take a number of hours to generate.

3.13.3.1 ATN

The ATN framework contains little that is specific to ATLAS. Knowledge of such details resides in user-provided scripts which are run by the ATN framework. Further, ATN was designed to be able to run other test frameworks such as QMtest, thereby leveraging the work invested in such products.

ATN is closely coupled to the nightly build system (NICOS). The results of the ATN framework are integrated into the NICOS status pages.

Domains currently being tested using ATN include Athena, database, detector systems (muon spectrometer, liquid argon calorimeter), simulation, reconstruction, trigger and kit validation.

3.13.3.2 KV

KV is a set of scripts used to test the ATLAS offline distribution kits and releases on AFS. It is easy to run, with quick turn-round which makes it a convenient tool for kit builders.

KV is currently used for kit validation and releases on AFS, and runs nightly under ATN and RTT.

3.13.3.3 RTT

The RTT adopts the point of view that ATLAS-specific information is to be understood by the framework, with the intention that the RTT should reduce the load on the ATLAS user to the minimum. To this end, the knowledge on how to run Athena, and other jobs, which provide the material which serves as input to the tests, is incorporated in the RTT. Further, common testing tasks are identified and placed in RTT libraries. As a result, the user does not need to write scripts to carry out the tasks of running standard jobs and standard tests, but interacts with the framework through configuration files. For those cases where the RTT-provided machinery is inadequate to fulfil users' needs, the user can provide a specialized test presented as a Python class. Under the RTT, information flows both from the user to the framework – the user specifies the test class and the arguments via the configuration file – and from the RTT towards the test instance: the RTT passes an information object when invoking the run method of the test instance.

Domains currently being tested using RTT include fast simulation, reconstruction, e-gamma, trigger, graphics, muon tracking, and QA.

3.13.4 Testing Future

The testing group is moving towards providing users with a coherent testing environment. The three independently developed test frameworks are run in different manners. Work is currently under way to harmonize this situation. Care will be taken to ensure that the frameworks continue to function during this process.

Testing will be organized by package. Each package will provide a single configuration file containing a section for each framework. Each framework will specify the possible content of the the corresponding configuration file section.

Tests which are common to the frameworks will be moved into libraries. To this end, it has been agreed to use the same scripting language. A common interface to the tests will make it possible to carry them out from any of the frameworks.

More work is needed to define the following:

- Description and storage of results;
- User interface to mine results;
- Grid integration.

3.13.5 Physics Validation

Physics validation is an investigation of the results produced by the offline software. The closest testing activity to physics validation is that of the large-scale integration tests. The activities differ in that validation is focused on what has gone wrong, whereas testing monitors deviations of software that has been established to be working correctly.

The final users of the Software and Computing infrastructure are ATLAS physicists who interact with this infrastructure every day to achieve their scientific goals. The quality of the software, both in terms of functionality and performance, needs to be monitored and feedback from the users brought to the attention of the computing management.

To facilitate the interaction between the software and physics communities, a Physics Validation forum was set up. The two main roles of this forum are:

- Inform the physics community of the latest news, e.g. software releases to use, problems that they may find and possible solutions. In this respect the Physics Validation forum provides a service to the physics working groups so that they can collect this information from a single source;
- Collect feedback from the users, identify possible problems and missing functionality. This aspect is complementary to the activity of the testing group, as users identify possible new problems that appear while trying to pursue a particular task. The participation of physicists from different working groups ensures that all areas of the ATLAS computing and software are stressed and used.

Physics validation meetings are held every two weeks and last for 1 to 2 hours. The meetings are informal and include presentations from people from different physics groups and provide a frequent information exchange to find common problems and solutions. A Physics validation coordinator organizes these meetings, and reports directly to the Physics coordinator. The same person also acts as a liaison for the physics requirement and feedback within the software management board.

3.14 Software Infrastructure

3.14.1 Introduction and Description

The ATLAS Software Infrastructure Team (SIT) is responsible for providing and maintaining the ATLAS software development environment. While the SIT is part of the offline software community, the SIT provides services to all communities using the offline software. The SIT work includes:

- Providing code management tools such as CVS, CMT, and the Tag Collector.[1]
- Managing the use of external (to ATLAS) software and providing interfaces to external code.
- Porting the software to a variety of platforms/compilers and providing debugging tools.

1. CVS, CMT and the Tag Collector are described in detail in Section 3.14.2, "Code Management".

- Providing tools and scripts for building releases.
- Producing software releases every night for testing and periodically for development, and production.
- Providing project builds to control dependencies and tools to monitor dependencies.
- Producing code distribution kits on supported platforms.
- Testing that the release code works properly. Testing is described in Section 3.13.
- Providing QA/QC (Quality Assurance and Quality Control) for the code.
- Providing documentation such as web and wiki pages, bug reporting and Doxygen.
- Providing user support, training, bug reporting, and a help mailing-list.
- Providing liaison with other ATLAS software communities (e.g. HLT and CTB).
- Maintaining the ATLAS mailing-lists and the disk space assigned to ATLAS.

Currently there are approximately 25 people representing about 9 FTEs of effort actively working on the SIT. Recent estimates indicate that SIT is about 5 FTEs short of the staffing level needed to fully perform the assigned tasks, and the SIT is therefore unable to cover all of its functions as it would wish.

3.14.2 Code Management

3.14.2.1 Concurrent Version System (CVS)

ATLAS uses the Concurrent Version System (CVS) [3-40] for the source-code repository and as the low-level management tool for the software code base. CVS supports the decomposition of the code base into a hierarchically organized set of *packages*, each containing a related set of files. Typically these files include the implementation and header files for C++ classes, together with documentation and configuration files. Typically a package maps onto a concrete component of the Athena architecture[1], or defines groupings of such components. CVS provides mechanisms for the checking out by one or more developers of a package or set of packages in order for them to modify, add, or remove files within the set of packages, and to commit such changes back to the repository so that they are available to other developers. CVS allows multiple developers to make such changes simultaneously, and if there are no inconsistencies will merge them together as they are committed. If conflicts are detected, CVS identifies them and requires that one developer resolve them before the merging is completed. Any package or set of packages may be tagged in such a way that this version may be recovered at any time in the future.

CVS implements a secure set of protocols in order to allow for check-outs and commits, based on a source-code repository that is managed by CERN IT [3-41] on a cluster of servers using a shared file system for performance and redundancy.

1. The Athena architecture and framework are described in Section 3.3, "The Athena Framework"

3.14.2.2 Code Management Tool (CMT)

The Code Management Tool (CMT) [3-42] is used as the basis for building consistent sets of packages and versions into a so-called ATLAS release, as the basis for supporting check-out and testing of packages during the software development process. Each package contains a configuration or *requirements* file that identifies which other packages it depends upon, the library, set of libraries and/or applications that it creates, and other configuration information that is used by other packages that depend upon this one, or are necessary in order to establish a consistent run-time environment. Standard *patterns* and *strategies* support common operations such as compilation of a set of C++ implementation files into a library, and provide support for different operating system and compiler combinations.

CMT allows dependencies to be examined, and circular dependencies to be detected.

3.14.2.3 Tag Collector

The Tag Collector is a web-based database application for release management. The tag collector allows code developers and librarians to select which versions (CVS tags) of each package are used to build the release. The first version was designed and implemented during the summer of 2001. The tool proved extremely successful, and a second, more powerful version known as Tag Collector II was introduced in the second quarter of 2005.

Both Tag Collector versions allow the preparation of builds of ATLAS software in a controlled fashion. The tool is interfaced with CVS, and also with CMT. Developers can interactively select the set of packages and their CVS tags to be included in a build, and the complete build commands are produced automatically. Other features are provided such as verification of container package CMT requirements files, and direct links to the package documentation. Tag Collector II provides a fine-grained management of user rights, some automated CVS tagging, and support for project-based builds which means that the total ATLAS offline software builds are replaced by several smaller independent builds. It is anticipated that this feature will enable a more efficient release cycle. Tag Collector II is based on the AMI generic database management software [3-43]. All Tag Collector commands can be accessed by the AMI web service.

3.14.2.4 Dependency Checking

The *checkreq* tool performs several internal consistency checks within each software package, with the main purpose of detecting any dependencies between packages which are either unnecessary or have been overlooked in defining the CMT requirements file for the package. Checkreq does a limited analysis of the requirements file of the package to collect the names of other packages referenced. It then does a limited analysis of the source files and header files contained in the package to assemble a list of files actually referenced there via include statements. Then a check is performed whether the included files, translated to package names, match the packages listed in the requirements file. Appropriate warnings are issued if there is no match. Additional checks are performed on the set of packages within a release, e.g. on consistency of package versions. The tool is implemented as a shell script and is maintained within the SIT. Checkreq is routinely run to check each referenced package during the nightly build procedures.

3.14.3 External Packages

The ATLAS offline software depends upon a set of externally developed and supplied software packages, including event generators and simulation tools and services and toolkits such as CLHEP [3-8]. In many cases the external packages depend upon each other and it is important that a consistent set of package versions is achieved. CMT supports access to external packages through the use of *glue* or *interface* packages, which identify the necessary package versions as well as access to the necessary package libraries and header files.

Since the offline software must operate in the high-level-trigger environment it is important that there is consistency in the package versions for external packages that are shared by both the offline and online/TDAQ systems. An explicit liaison procedure has been put into place to ensure such consistency.

The categories of external packages that are used by the ATLAS offline software are:

- Packages developed by the LCG Applications Area (see Section 3.4, "Data Access Model") and other external packages that these themselves depend upon.
- Monte Carlo event generators (see Section 3.8.2, "Generators").
- Packages common to both the online and offline software.
- Java support packages.
- Miscellaneous.

3.14.4 Platforms and Compilers

Since January 2005 the default Linux version at CERN is Scientific Linux CERN 3 (SLC3) [3-44]. It is a CERN-customized Linux distribution built on top of a common base platform, Scientific Linux, which is in turn built from freely available Red Hat Enterprise Linux 3 sources by a joint Fermilab and CERN effort. SLC3 is built to integrate into the CERN computing environment but it is not a site-specific product. Over the period between January and June 2005, SLC3 replaced Red Hat Linux 7.3 on essentially all machines in the CERN computer centre. The introduction of SLC3 required the software building team to expend considerable effort to overcome significant compatibility issues between the build procedure, the software, and SLC3.

In addition to support for SLC3, ATLAS has and continues to make some efforts to support platforms other than SLC3. The ATLAS software has been demonstrated to run acceptably on AMD Opteron machines in 32-bit mode. Work is under way both within LCG and ATLAS to support Opteron running in 64-bit mode. Previously LCG had provided support for the LCG-written external packages used by ATLAS on the Intel Itanium (IA-64) but this effort is currently dormant because of the lack of acceptance for the Itanium processor. ATLAS has also put some effort into porting the ATLAS software to Macintosh OSX but while there is considerable interest within the software developer community, lack of available manpower has limited progress on an OSX port. The SIT is also actively working on ways to make building the ATLAS software and the associated externals on different Linux distributions easier (currently only the SLC3 variant of Red Hat Enterprise Linux 3.0 is officially supported). The plan is to collaborate with various groups that have the need to run the software on clusters with other Linux distributions installed.

Distcc [3-45] is a fast, free distributed compiler that the SIT is investigating to speed the compilation of ATLAS software. The offline software now takes approximately one day to build on one of the best machines available at CERN. Distcc distributes builds of C, C++, Objective C and Objective C++ code across several machines on a network. When the software is properly configured, distcc generates the same results as a local build, is simple to install and use, and is much faster than a local compile. It does not require all machines to share a file system, have synchronized clocks, or to have the same libraries or header files installed. ATLAS has dedicated access to five distcc server machines.

3.14.5 Releases and Release Strategy

A hierarchy of release builds is used to ensure rapid feedback of package integration problems and as testbeds for testing and validation. The hierarchy is:

- *Nightly Releases*. The complete ATLAS offline release is built every night, several such builds being made available before the oldest one is over-written in a cyclic arrangement. Seven such releases are maintained in the main development branch, but the number on bug-fix branches (see later) is typically less because of disk space constraints. Similarly, not all combinations of platforms, compilers, and optimization and debug levels are built.

  An incremental build strategy is used to minimize the time it takes to perform the builds, each build for a single platform taking place in a single disk location, only updated and new packages (and packages that depend upon those) being built.

  Nightly builds are fully automatic, and there is no human intervention if the build fails, either because of a package configuration problem, or an infrastructure problem (e.g. AFS error).

  The list of packages and their associated tags for the nightly builds, as well as the other build types described below, is determined by the ATLAS Tag Collector described in Section 3.14.2.3, "Tag Collector".

  NICOS is the NIghtly COntrol System that manages these multi-platform nightly builds based on the recent versions of the software. NICOS performs nightly build processes in distinct steps: compilation, tests (including quality assurance, unit, and integration), automatic problem analysis, and creation of web pages with build summaries. Developers are automatically notified about problems with their packages via e-mail.

- *Developer Releases.* Developer releases occur approximately every three weeks, driven by the set of packages and tags in the Tag Collector. In contrast to the nightly releases, the information from container packages is used, and consistency in the packages and tags is enforced between the set derived directly from the Tag Collector, and the set derived from the container packages.

  Developer releases are subjected to limited QA testing and a distribution kit (see Section 3.14.6, "Code Distribution") is created in order to allow them to be installed at remote sites.

- *Production Releases*. Production releases occur approximately every 4–6 months, normally (but not necessarily) being targeted towards high-level milestones such as data challenges, physics workshops or test-beam operation. The build procedure is essentially identical to that of the developer releases, the major differences being that there is more flexibility in delaying the release build if required functionality is missing, and in the amount of QA testing that is carried out. Acceptance of the release is dependent on a positive report card

from the relevant communities, typically the Physics Coordination, or High-Level Trigger etc.

A distribution kit (see Section 3.14.6, "Code Distribution") is built for each production release, which allows it to be installed and utilized at remote sites without AFS access.

- *Bug-Fix Releases*. Bug-fix releases can be associated with production releases if subsequent extended testing uncovers problems of a severity that the production release is deemed unusable for serious production. In this case, a bug-fix release (or set of such releases) may be created. These are supported in the Tag Collector, being associated with their base release. In general, only specific packages and tags are accepted, under the authority of the Release Coordinator.

  A distribution kit is built for each bug-fix release.

3.14.5.1 Project Releases

From its inception ATLAS offline software releases have been made for the entire code base. This has put stress on the release-build hardware, and also compromises the robustness of the software to changes in the core packages. A reorganization is therefore under way, with the goal being to split the offline software into distinct *projects*, each project possibly depending upon other projects, but having their own release cycle and support tools (e.g. Tag Collector, release coordinator). Packages within each project can only depend upon other packages within the same project or from a lower project.

The tentative project decomposition is:

- *AtlasCore*. This is the core set of packages that are common to all other projects.
- *AtlasConditions*. This is the set of packages that deal with the geometry, misalignment and calibrations for all the detector sub-systems.
- *AtlasEvent*. This is the set of packages that deal with the Event Data Model.
- *AtlasSimulation*. These comprise the event generators and simulation packages.
- *AtlasReconstruction*. These packages cover not only the event reconstruction, but also online monitoring.
- *AtlasAnalysis*. Packages associated with physics analysis.
- *AtlasTrigger*. Packages associated with the high-level trigger.

The assignment of packages to these projects is under way, with the goal that these project-based builds will be in production in September 2005.

3.14.5.2 Release Coordinator

The Release Coordinator has overall responsibility for the release builds, having a term of duty that extends over a single production release cycle (i.e. from one production release until the next one). They have ultimate authority over whether to delay a release or reject late submissions, etc.

The Release Coordinator makes sure the (developer and production) release deliverables are met, by staying in contact with the developers and coordinators involved. Given the importance of functionality of the nightly builds and developer releases for the success of a production re-

lease, the Release Coordinator regularly monitors the functionality of the nightly release to make sure that no serious problem remains unfixed. The Release Coordinator coordinates major changes to minimize disruption and ensures that developers are informed about major changes, and about the status/usability of developer releases.

3.14.6 Code Distribution

The architecture of the code distribution for the ATLAS software is based on a combination of the CMT code management tool and the Pacman [3-46] code packaging tool. A set of shell scripts that are part of the librarian toolkit performs the appropriate queries to the CMT knowledge base provided by the package authors in the form of package requirements files, and construct a Pacman cache from it. Users wishing to install the ATLAS software can simply use Pacman to download and install it on their machines. Although the system is in production (it has been greatly exploited for all recent data production activities), several elements are still continuously evolving and improving, since several aspects of the functionality are not completely implemented such as incrementally building the kits, increasing traceability, etc.

3.14.7 Quality Assurance and Quality Control (QA/QC)

The essential difference between QA and QC is that the former describes an approach to software development, whereas the latter implies that development is subject to a series of tests measuring the software quality. The QA/QC coordinators pursue a two-pronged approach providing support for voluntary peer reviews of a technical nature, and also support for a tool which checks the compliance of code with ATLAS C++ coding rules.

Until 2005, the number of technical reviews within ATLAS software has been relatively small. In the first half of 2005 ATLAS software management has adopted the QA/QC recommendations for review procedure in a series of 10 non-technical reviews. This experience has had a positive effect in that it has encouraged sub-groups to organize their own technical reviews, following the same procedure.

ATLAS has adopted the “RuleChecker” code parser [3-47]. The initial contract with the vendor was terminated before it was technically possible to run the tool over all the ATLAS release. Since March 2005 it has become possible to do this, using the RTT tool, allowing a better view of the ensemble of the results. The tool was used with some success by the database group during their January 2005 documentation drive. Some RuleChecker parsing bugs have been exposed, which are rapidly fixed by the developers. However, systematic use of the tool has also exposed weaknesses and inconsistencies in the coding rules, and any modification of the tool to accommodate changes to these rules will require a new contract with the vendor.

Quality Control is discussed in detail in Section 3.13, "Testing and Validation".

3.14.8 Documentation

At present the documentation of ATLAS software exists at multiple places and in various forms. The principal sites for access to documentation are:

- The standard web pages [3-48]: They are the central start point for finding documentation, including the items listed below.

- The Doxygen code documentation [3-49]: It is extracted from comments in the code. Additional pages, especially the 'mainpage', provide introduction to the code and navigation through the classes.
- The TWiki web pages [3-50]: They have become a very popular way of documentation, communication and developing ideas within a group.
- The Workbook [3-51]: It is part of the TWiki web pages, but contributions are more tightly controlled. It will become the portal for new-comers to ATLAS computing, but will also provide pointers for more specialized information.
- Software notes on CERN CDS or in the ATLAS CVS repository;
- Tutorials of introductory courses with hands-on exercises.

An effort is under way to assess, upgrade and streamline the existing documentation [3-52]. Special attention is paid to documentation intended for users who are new to ATLAS. The example of BaBar is being followed, which has very good experience with documentation for newcomers in the workbook style.

Recommendations for web pages and usage of Doxygen will ensure better control of correctness and timely maintenance of the information. Each page must list at least the responsible person and the date of the last significant update.

Documentation is a common effort. Once the initial layout and the definition of rules and templates are available, the contributions of experienced developers, and the review by experienced and new users must be encouraged. Communication and cooperation between the software and physics communities, and the developers and users is needed to keep the documentation up-to-date and useful, and to avoid duplication of effort.

3.14.9 User Support

ATLAS uses the Savannah [3-53] bug tracking system for reporting problems or following the progress of “tasks”. Problems can be posted to Savannah anonymously or under a registered username if a user chooses to log in. Currently ATLAS has 36 bug tracking groups (projects) in Savannah. There is one group (ATLAS Bugs), which is a general group intended for use by users who do not know to which group their bug should be posted. The manager of the bug group assigns each bug to a member of the group who then becomes responsible for following the bug and seeing that it is fixed. Every action within Savannah generates an automatic email to all of the people working on the bug, the group managers, and the person submitting the bug (unless the submission was anonymous).

The “atlas-sw-help” mailing list was originally set up to reduce traffic on the “atlas-sw-developers” list. Membership was meant to be limited just to “experts” who would feel duty-bound to respond to requests, which would often be from beginning users. However, for various reasons this mechanism has not been entirely successful and the SIT is currently considering the future of this list. It seems likely that membership will be reset to a reduced number of people, and that a moderator will be put in place. Even though many people do not feel that a mailing list is the best solution for this kind of user support, all attempts to replace the mailing list mechanism by a forum or news group have failed.

Another aspect of user support is managing disk space requests and allocations, both on behalf of ATLAS users, and for ATLAS operations. A general support mailing list (“atlas-support”) is

available for these requests. Disk space management is particularly important for the ATLAS releases. A policy has been established to control the deletion of obsolete releases and to allow ample time for physicists to migrate to newer and more functional releases. Releases are archived prior to deletion.

3.14.10 HLT Coordination

Figure 3-15 shows the dependencies between the online, high-level trigger, offline, and external software. This tight coupling requires good coordination to ensure that consistent package versions are maintained. Representatives from the online and HLT communities are invited to the regularly scheduled SIT meetings, and to the ATLAS computing-management meetings to ensure good communication for this and other coordination issues.

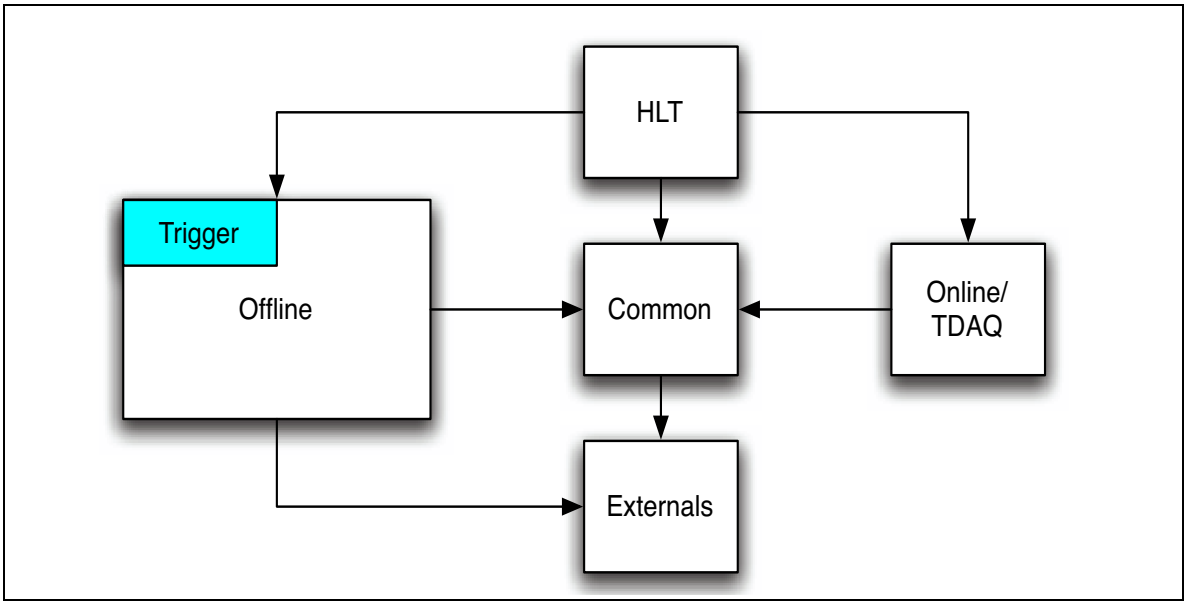

Figure 3-15 Online, High-Level Trigger and Offline dependencies

3.14.11 Mailing Lists

Use of mailing lists is an essential part of the communication between ATLAS collaborators. Many lists are used, targeted at different user and developer communities. The CERN SIMBA2 mailing list management system is used to control access, both for subscribing to receive email notifications, and for authority to be able to post to specific lists. The lists themselves are archived and provide a web interface.

3.15 Integration of LCG Application Area Products

ATLAS is both a contributor to, and client of, products from the LCG Application Area. The major products that are used within ATLAS are:

- SEAL;
- POOL;

- PI;
- Simulation components;
- Components and services supported by the SPI project;
- The LCGCMT glue packages.

3.15.1 SEAL

The SEAL project [3-54] provides the software infrastructure, basic frameworks, libraries and tools that are common among the LHC experiments. The project addresses the selection, integration, development, and support of foundation and utility class libraries. These utilities cover a broad range of unrelated functionalities and it is essentially impossible to find a unique optimum provider for all of them. They should be developed or adapted as the need arises. In addition to these foundation and utility libraries, the project should develop a coherent set of basic framework services to facilitate the integration of LCG and non-LCG software to build coherent applications. SEAL contains the following major components:

- Foundation and Utility Libraries. These include basic utility and system isolation libraries, utility classes for stream oriented I/O and support for I/O compression.
- Math Libraries. These provide physics vector and linear algebra libraries, numerical algorithms (integration, differentiation, minimization, etc.) libraries, random-number generators and distributions.
- Dictionary. The LCG Dictionary is an attempt to add introspection (reflection) functionality to the standard C++ language. The Reflection package provides the reflection API to the dictionary information. The ReflectionBuilder package is used to build at run-time the dictionary information in memory based on parsing of the C++ header files. A new version of the Reflection library called Reflex has been released in parallel and is being tested. This new version is closer to the ISO/IEC 14882 standard and will replace the Reflection and ReflectionBuilder libraries in the future.
- Component Model and Plug-in Manager. This provides services for managing, querying, dynamic loading and unloading of plug-in components, and application bootstrapping (initialization), and the definition of an object lifetime strategy and support tools.
- Framework and Services. This extends the component model by providing an infrastructure for component development and a set of concrete services.
- Python Services. These provide Python language bindings for the standard services and utilities developed within SEAL, scripting support for applications, and Python bindings to ROOT services and utilities.

ATLAS makes heavy use of the Dictionary and Python services, and has begun incorporating aspects of the component model into its software. A migration from the existing Athena component model to the SEAL one is planned, although this has been deferred until after the reorganization of the LCG Application Area and a clarification of the SEAL/ROOT merger. Similarly, a migration towards more extensive use of the mathematical libraries has been deferred pending a similar evaluation.

3.15.2 POOL

The POOL project [3-55] has been created to implement a common persistency framework for the LHC Computing Grid (LCG) application area. POOL can store multi-petabyte experiment data and metadata in a distributed and Grid-enabled way. The project follows a hybrid approach combining C++ Object streaming technology, such as ROOT I/O [3-56], for the bulk data with a transaction-safe relational database (RDBMS) store, such as MySQL [3-57]. POOL uses a component approach based on the SEAL component model providing navigational access to distributed data without exposing details of the particular storage technology.

The use of POOL within the ATLAS software environment is described in more detail in Chapter 4, "Databases and Data Management".

3.15.3 PI

The Physicist Interface (PI) project [3-59] encompasses the interfaces and tools by which physicists will directly use the software. The project provides:

- Analysis Services. These provide extensions to the AIDA [3-60] analysis APIs, the implementation of some of them using ROOT, and an extended AIDA API to SEAL and POOL services.
- Analysis Environment. This provides a framework for interactive applications together with a set of core services based on POOL and SEAL.
- Visualization support.

ATLAS uses the ROOT implementation of the AIDA histogram API provided by the PI project.

3.15.4 Simulation Components

The simulation project [3-61] of the LCG Applications Area encompasses common work among the LHC experiments on the development of a simulation framework and infrastructure for physics validation studies, CERN and LHC participation in Monte Carlo generator services, Geant4, Fluka and Garfield. Its work is guided by the reports of the simulation, the Monte Carlo generators and the detector description RTAGs. The relevant sub-projects are:

- Simulation Framework. The general task of this sub-project is to provide flexible infrastructure and tools for the development, validation and usage of Monte Carlo simulation applications.
- Geant4. This sub-project encompasses the effort undertaken in CERN PH/SFT group in maintaining, supporting and developing further the Geant4 simulation toolkit. The work plans prepared are driven by the requirements for support of LHC production usage, maintenance of a number of toolkit components, collection of new experiment requirements and creation of new functionality.

  The work is carried out in close collaboration with Geant4 colleagues in many institutions around the world, working to address the requirements of these and other user communities and contributing to the development, support and maintenance of the complementary set of toolkit components. Together, in the Geant4 Collaboration, this sub-project provides the releases of the Geant4 toolkit for all users, including users in the LHC exper-

iments, other HEP experiments and facilities, and for applications in fields as diverse as medicine and space.

- Physics Validation. The work of this project includes:
  - Understanding the impact on LHC physics of uncertainties or inadequacies of the simulation on the quality of the physics results;
  - Providing feedback based on the test-beam activities of the LHC experiments;
  - Performing studies based on the actual detector geometries;
  - Performing electromagnetic studies and create optimized physics lists;
  - For hadronic studies looking at the impact of tracking, calorimeters and backgrounds;
  - Evaluating results from outside of the LHC;
  - Validating the performance and functionality of the simulation environment.

  ATLAS has contributed significantly to the physics validation project as a result of feedback from a variety of test beams. This feedback continues since the analysis of the combined test-beam results is still in progress.
- Generator Services. The mandate of this sub-project is to collaborate with Monte Carlo (MC) generator authors and with LHC experiments in order to prepare validated LCG-compliant code for both the theoretical and experimental communities at the LHC, sharing the user-support duties, providing assistance for the development of the new object-oriented generators, and guaranteeing the maintenance of the older packages on the LCG-supported platforms. Four different sub-projects have been defined:
  - Generator services library and new Object-Oriented MC generators;
  - Storage, event interfaces and particle services;
  - Shared generator-level event files: production and MC database;
  - Validation and tuning.

  Wherever possible ATLAS uses versions of event generators that are supported and validated by this SEAL project. ATLAS performs significant validation and the results are propagated back so as to result in improvements to the generators and configuration information.

3.15.5 SPI Components

ATLAS uses the following SPI [3-62] components:

- Savannah. This is a project support portal, but which is used within ATLAS primarily as a bug-tracking system.
- Testing tools.

3.15.6 LCGCMT

The LCGCMT glue packages provide a mechanism whereby consistency across multiple external software package versions can be achieved. LCGCMT is used within ATLAS to ensure that

the ATLAS offline uses a self-consistent set of external packages with the LCG Application Area software.

3.16 References

3-1 ATLAS Collaboration, *ATLAS High-Level Trigger, Data Acquisition and Controls Technical Design Report*, CERN-LHCC-2003-022 (2003)

3-2 Athena Developers Guide,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/General/Documentation/AthenaDeveloperGuide-8.0.0-draft.pdf

3-3 G. Barrand *et al.*, *Gaudi - A Software Architecture and Framework for building HEP Data Processing Applications*, International Conference on Computing in High Energy Physics (CHEP) (2000)

3-4 LHCb,
http://lhcb.web.cern.ch/lhcb/

3-5 Particle Data Group,
http://pdg.lbl.gov/

3-6 HLT - Histogram Template Library,
http://wwwasd.web.cern.ch/wwwasd/lhc++/HTL/

3-7 ROOT,
http://root.cern.ch/

3-8 CLHEP,
http://wwwasd.web.cern.ch/wwwasd/lhc++/clhep/

3-9 Python,
http://www.python.org/

3-10 STL,
http://www.sgi.com/tech/stl/

3-11 The Geant4 Collaboration (S. Agostinelli *et al.*), *Geant4 - A Simulation Toolkit*, Nuclear Instruments and Methods in Physics Research, NIM A 506 (2003), 250-303

3-12 Adele Rimoldi *et al.*, *The simulation for the ATLAS experiment: present status and outlook*, International Conference on Computing in High Energy Physics (CHEP), Interlaken (2004)

3-13 S. Spagnolo *et al.*, *The Description of the ATLAS Detector*, International Conference on Computing in High Energy Physics (CHEP), Interlaken (2004)

3-14 S. Goldfarb and A. Schaffer (editors), *Definition of Offline Readout Identifiers for the ATLAS Detector*, ATLAS Internal Note, ATL-SOFT-2001-004 (2001)

3-15 DALI,
http://aleph.web.cern.ch/aleph/dali/

3-16 M. Virchaux, D. Pomarede, *The PERSINT Manual*, ATLAS Internal Note, ATL-SOFT-2001-003 (2001)

3-17 ATLAS Muon Detector Description,
http://muondoc.home.cern.ch/muondoc/Software/DetectorDescription/

3-18 D. Pomarede *et al.*, *Graphics of Level-1 Objects in the ATLAS Barrel Muon Spectrometer Trigger System*, ATLAS Internal Note, ATL-DAQ-2002-001 (2001)

3-19 ATLAS Collaboration, *ATLAS Detector and Physics Performance Technical Design Report,* CERN-LHCC-99-14 and CERN-LHCC-99-15 (1999)

3-20 ATLAS Muon Collaboration, *ATLAS Muon Spectrometer Technical Design Report*, CERN-LHCC-97-22 (1997)

3-21 *A summary of the ATLAS MDT Calibration Model*, ATLAS Internal Note in preparation

3-22 R. Hawkings and F. Gianotti, *ATLAS detector calibration model and preliminary subdetector requirements*,
http://atlas.web.cern.ch/Atlas/GROUPS/DATABASE/project/calib/doc/calib_req.pdf

3-23 Genser,
http://lcgapp.cern.ch/project/simu/generator/

3-24 E. Richter-Was, D. Froidevaux, L. Poggioli, *ATLFAST 2.0 - a fast simulation package for ATLAS*, ATLAS Internal Note, ATLAS-PHYS-98-131 (1998)

3-25 ATLAS Collaboration, *ATLAS Letter of Intent for a General-Purpose pp Experiment at the Large Hadron Collider at CERN*, CERN/LHCC/92-4 (1992)

3-26 Geant4,
http://wwwasd.web.cern.ch/wwwasd/geant4/geant4.html

3-27 D. Barberis, G. Polesello and A. Rimoldi, *Strategy for the Transition from Geant3 to Geant4 in ATLAS*, ATLAS Internal Note, ATL-SOFT-2003-013 (2003)

3-28 A. Rimoldi *et al.*, *The Simulation of the ATLAS Experiment: Present Status and Outlook*, ATLAS Internal Note, ATL-SOFT-2004-004 (2004)

3-29 D. Costanzo *et al.*, *Validation of the Geant4-Based Full Simulation Program for the ATLAS Detector: An Overview of Performance and Robustness*, ATLAS Internal Note, ATL-SOFT-PUB-2005-002 (2005)

3-30 G. Blazey *et al.*, *Run II Jet Physics*, hep-ex/0005012v2 (2000)

3-31 J. Butterworth *et al.*,
see http://jetweb.hep.ucl.ac.uk/ktjet/index.html

3-32 V. Boisvert *et al.*, *Final Report of the ATLAS Reconstruction Task Force*, ATLAS Internal Note, ATL-SOFT-2003-010 (2003)

3-33 PAW,
http://wwwasd.web.cern.ch/wwwasd/paw/

3-34 JAS,
http://www-sldnt.slac.stanford.edu/jas/

3-35 UCL Workshop Summary Report,
http://www.usatlas.bnl.gov/PAT/ucl_workshop_summary.pdf

3-36 K. Assamagan *et al.*, *Final Report of the ATLAS AOD/ESD Definition Task Force*, ATLAS Internal Note, ATL-SOFT-2004-006 (2004)

3-37 Physics Analysis Tools CVS Repository,
http://reserve02.usatlas.bnl.gov/lxr/source/atlas/PhysicsAnalysis/

3-38 Physics Analysis Tools web site,
http://www.usatlas.bnl.gov/PAT/

3-39 M. Boonekamp *et al.*, *Cosmic Ray, Beam-Halo and Beam-Gas Rate Studies for ATLAS Commissioning*, ATLAS Internal Note, ATLAS-GEN-2004-001 (2004)

3-40 CVS,
http://www.gnu.org/software/cvs/

3-41 CERN Central CVS Service,
http://cvs.web.cern.ch/cvs/

3-42 CMT,
http://www.cmtsite.org/

3-43 AMI,
http://atlasbkk1.in2p3.fr:8180/AMI/

3-44 Scientific Linux 3,
http://linux.web.cern.ch/linux/scientific3.

3-45 Distcc,
http://distcc.samba.org/

3-46 Pacman,
http://physics.bu.edu/~youssef/pacman/

3-47 RuleChecker, IRST, Trento, Italy.

3-48 ATLAS Computing web site,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/

3-49 Doxygen Code Documentation,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/Release/Doxygen

3-50 ATLAS TWiki,
https://uimon.cern.ch/twiki/bin/view/Atlas/WebHome

3-51 ATLAS Workbook,
https://uimon.cern.ch/twiki/bin/view/Atlas/WorkBook

3-52 T. Kozanecki *et al.*, ATLAS documentation Task Force,
http://hansl.home.cern.ch/hansl/atlas/documentation/documentationTF.pdf

3-53 Savannah,
https://savannah.cern.ch

3-54 SEAL,
http://seal.web.cern.ch/seal/

3-55 POOL,
http://lcgapp.cern.ch/project/persist/

3-56 ROOT I/O,
http://root.cern.ch/root/InputOutput.html

3-57 MySQL,
http://www.mysql.com/

3-58 LCG Blueprint RTAG,
http://lcgapp.cern.ch/project/blueprint/

3-59 PI,
http://lcgapp.cern.ch/project/pi/

3-60 AIDA,
http://aida.freehep.org/index.thtml

3-61 LCG Simulation Project,
http://lcgapp.cern.ch/project/simu/

3-62 SPI,
http://lcgapp.cern.ch/project/spi/

4 Databases and Data Management

4.1 Introduction

One of the principal challenges for ATLAS computing is to develop and operate a data storage and management infrastructure able to meet the demands of a yearly data volume of O(10 PB) utilized by data processing and analysis activities spread around the world. The ATLAS Computing Model discussed in Chapter 2 establishes the environment and operational requirements that ATLAS data handling systems must support, and, together with the operational experience gained to date in test beams and data challenges, provides the primary guidance for the development of the systems described below.

In May 2004 the ATLAS Databases and Data Management Project (DB Project) was established to lead and coordinate ATLAS activities in these areas, with a scope encompassing technical databases (detector production, installation and survey data), detector geometry, online/TDAQ databases, conditions databases (online and offline), event data, offline processing configuration and book-keeping, distributed data management, and distributed database and data management services. The project is responsible for ensuring the coherent development, integration and operational capability of the distributed database and data management software and infrastructure for ATLAS across these areas.

This chapter describes the work completed, under way and planned within the DB Project to meet the data-handling needs of ATLAS at startup and beyond.

4.2 Approach and Architecture

There are two broad categories of data storage in ATLAS: file-based data and database-resident data or, more specifically, relational-database-resident data. The two storage approaches are complementary and are used in appropriate contexts in ATLAS. File storage is used for bulky data such as event data and large volume conditions data; for contexts in which the remote connectivity (usually) implied by database storage is not reliably available; and generally for cases where simple, lightweight storage is adequate. Database storage is used where concurrent writes and transactional consistency are required; where data handling is inherently distributed, typically with centralized writers and distributed readers; where indexing and rapid querying across moderate data volumes is required; and where structured archival storage and query-based retrieval is required.

While these two storage categories cover ATLAS needs very well, their mapping to the software environments of ATLAS is not obvious or trivial, and achieving a low-impedance mapping of programmatic data access to data storage has engendered much work in recent years, with some success. ATLAS software environments both online and offline are primarily object-based, with C++ the most widely used language and the standard in offline (used for the offline Athena framework).

File-based storage of C++ objects is made possible through the use of ROOT I/O, which provides high-performance and highly-scalable object serialization to self-describing, schema-evolvable, random-access files. This capability relies on the foundation of a C++ object diction-

ary, providing the object introspection needed for automatic serialization. ATLAS employs ROOT I/O via the POOL persistency framework, which isolates applications from the storage technology used (a capability utilized in relational storage as discussed below), and provides the added value of data navigation at the object level with navigational support extending Grid-wide through an interface to file-catalogue services, which may be Grid-based. POOL supports persistent object references, which record not only in-file object location but file identity (via a unique identifier, the GUID) with support for automatic navigation to the required physical file through a catalogue lookup (GUID dereferenced to a physical file location). Thus ROOT provides the bridge from our transient objects to highly efficient file-based storage, and POOL provides the further bridge from these files to a catalogued, navigable, distributed file-management infrastructure.

For database storage, ATLAS and LHC experience over many years has narrowed the technologies of interest to SQL-based relational databases but has shown the value of flexibility in the choice of relational database engine (vendor). While Oracle is CERN IT's choice for their large-scale DB service, MySQL has emerged as the preferred engine for installations outside of institutional IT departments, and SQLite combines SQL relational DB with local file-based storage which is very attractive in some contexts. Oracle, and specifically CERN IT's Physics Database Service, will be the key central database service acting as archival repository for ATLAS DB-resident data and often the 'write master' with replication from CERN Oracle to other technologies for distributed read-only use. Vendor neutrality in the DB interface (with implemented support for the three mentioned vendors) has been addressed through the development of the Relational Access Layer (RAL) within the POOL project to provide performance-optimized, vendor-neutral, C++-based DB access, with a transparent mapping to DB schema such that RAL-accessed DBs can also be accessed through conventional DB tools. ATLAS utilizes RAL both directly and through layered services for DB storage via C++ with either Oracle, MySQL or SQLite back-end engines.

The RAL layer is used both directly by simple applications and indirectly as the foundation of higher-level DB-based storage services provided by POOL and by the conditions database software COOL. POOL offers RAL-based relational DB storage via the same StorageManager interface as the complementary ROOT file storage. Object-relational POOL provides for the storage of C++ objects of moderate complexity in relational DBs through exactly the same interfaces as ROOT file-based storage. POOL (and ATLAS) uses this capability to support the storage of event collections (tag databases) as either ROOT files (TTree based, usable directly in the ROOT environment for TTree based analysis) or DB-based collections; both these storage modes are very useful in the context of event collections. Object-relational POOL is also being deployed to store conditions data in a relational DB, via an interface that allows easy switching between POOL file-based and DB-based conditions-data storage as conditions DB storage and access is optimized.

COOL (developed in a collaboration between LCG AA and ATLAS) is another DB-based storage service layered over RAL and is the basis for ATLAS conditions data storage. It provides for interval-of-validity (IOV) based storage and retrieval of conditions-data objects, which may be stored by a variety of means: 'inline' with the IOV information for small simple data, in another user-defined DB table, in a POOL object stored via either object-relational POOL or POOL ROOT files, or in a file of arbitrary type (referenced and accessed via the services of the distributed data management system).

All storage systems used by ATLAS are interfaced to offline software via the Athena service and persistency conversion infrastructure. Developments are under way to make ATLAS data stores accessible also from a native ROOT environment, anticipating its use for analysis. Storage sys-

tems important to online, in particular COOL, are supported for online via dedicated interfaces to online data and information systems.

The bridge from ATLAS file and database-storage systems to their full availability at scale in the distributed ATLAS computing environment is the least developed area at the time of writing, and is under intensive development following strategic and architectural paths that are for the most part clearly established. The key here is, on the one hand, to extend the foundational tools of ATLAS data storage, POOL, RAL, COOL and ROOT, outwards over the Grid through their integration with distributed computing tools and services, and, on the other hand, to build an ATLAS distributed-data management system proper, to support distribution and discovery of ATLAS data across the collaboration. The end objective is to ensure that all ATLAS physicists wherever they are located have equal access to all ATLAS data (extending to support for small institutes and personal computers), within the constraints of available resources. A vital element is close collaboration between ATLAS and a strong LCG Distributed Deployment of Databases (3D) project. In the context of the 3D project the infrastructure and service relationships to support databases needed throughout the computing tiers and data replication between them are being developed. 3D is also developing the software systems to support authentication, monitoring at both server and client sides, server selection supporting load balancing and failover, and multi-tiered data-access systems to provide scalable retrieval of DB-resident data. These software systems are being integrated with or interfaced to RAL, COOL and POOL. Another vital element is the work under way to expand ROOT's support for the distributed grid environment. As the foundation for file-based data storage by POOL, COOL, etc., ROOT's interfaces to distributed storage services can be leveraged directly. ROOT provides interfaces to distributed storage tools such as dCache, xrootd, GFAL, and soon SRM. Grid authentication tools developed by ROOT are being integrated at higher levels by POOL and 3D.

The distributed data-management system, which is heavily based on relational databases, is being developed to leverage much of the same work: POOL file catalogue interfaces and implementations, RAL, the 3D distribution/replication infrastructure. It also will leverage mature components of Grid middleware infrastructure, such as authentication services and file-transport tools.

The following sections address the specifics of database and data management work in the various domains, and they describe how the architectural approaches just outlined are developed and applied in those domains.

4.3 Technical Data

4.3.1 Technical Coordination Databases

The technical coordination database area gathers together all the databases used for the production and installation of the ATLAS detector hardware and infrastructure. This data is used both by ATLAS technical coordination (TC) and sub-detectors during the integration and installation phases of the experiment, and will in many cases be used to initialize other databases, e.g. the online configuration and conditions databases. Some of the data will be required on a long-term continuing basis, e.g. to track the movement of equipment in and out of the experimental cavern throughout the life of ATLAS, in particular to satisfy the radiation safety requirements of the legal authorities (INB regulations).

4.3.2 Detector Production Data

Many separate databases have been used during the production phase of the ATLAS experiment, to keep track of detector parts, assemblies, test and quality assurance data. Much of this data is spread throughout the collaboration, using a variety of tools ranging from MS Access, MySQL and Oracle databases, to text files, reports and unstructured information. An effort is under way to migrate as much of this data as possible to the CERN Oracle physics database services, to ensure its long-term availability and accessibility. Sub-detector groups are being encouraged to create their own database schema on the CERN physics database server, with data structures appropriate to each individual data and application, rather than trying to organize all the various types of data in the same way. So far, data has been stored from a number of ATLAS sub-detectors, including mechanical characterization and electronic test data for the LAr calorimeter, and validation and certification data from several types of muon chambers.

To assist sub-detectors and TC in accessing this varied data, a generic database search interface, known as 'Glance' has been developed. This system reads the database schema, and automatically produces search interfaces based on the table structure, which can be customized using XML files. This allows tailored search interfaces to be quickly set up for any set of ATLAS production data, without the need for the development of dedicated interface software. This system has been used both for production data, and the ATLAS integration and cabling databases discussed below.

Some sub-detectors have set up sophisticated database systems of their own, e.g. the SCT production database hosted on Oracle servers at the University of Geneva, used by both SCT and pixel detectors for all their module production and characterization data. In such cases, where the long-term data integrity and support is already ensured by outside institutes, no migration of production data to CERN is foreseen.

4.3.3 Installation Data

Many thousands of separate detector and infrastructure parts are being installed in the ATLAS cavern and counting rooms, all of which need to be tracked and accounted for, especially in view of the INB safety regulations. For this purpose, ATLAS is making heavy use of the MTF (Manufacturing and Test Folder) system based on web interfaces, MP5 and the CERN Oracle database servers. This system has been developed at CERN, and is also used for tracking the installation of the LHC machine. Within MTF, the ATLAS detector is represented as a tree structure, with branches for each sub-detector and the individual leaves being pieces of the detector which are installed individually - e.g. a detector module or electronics crate. Each such piece has an 'equipment passport' giving its unique identifier, equipment code, physical properties (dimensions and weight), current and final locations, and links to other information such as specification documents or drawings. MTF also allows other information to be associated with each object, such as links to documents in the CERN engineering-data management system (EDMS), and in particular information on the current status of the object and tests it has undergone. All ATLAS components that are installed underground will be entered into MTF, and will be identified by barcode stickers giving the unique equipment identifier.

An important part of the installation database effort is concerned with managing the electronics racks and cables around the detector. This data is stored in the ATLAS integration database, which is maintained on the CERN Oracle physics database servers, and contains details on the location and contents of racks and the crates inside them. This database is accessed using the

Rack Wizard, a Java-based graphical tool developed by CMS and in use by several LHC experiments [4-1]. The Rack Wizard is being used for defining the rack contents, planning for infrastructure needs (e.g. power and cooling requirements) and tracking installation. The system is currently being extended to cover racks and patch panels on the detector itself, and to provide graphical views of their location and contents.

A cabling database has recently been implemented in the context of the integration database framework. This database stores information on the purpose, physical characteristics, routing and status of each cable, together with the associated connectors and cable trays. It is currently being used to plan and track the cable installation, and an interface to the Rack Wizard, allowing the connection of cables to electronics boards and crates, is under development. Bulk data import and export is performed by means of Excel spreadsheets, which are prepared by sub-detector cabling experts, and search/update interfaces are under development, including access via cable-label barcode readers, which will be used to track cable installation and testing in the ATLAS cavern and counting rooms.

The cable database will also be used to track the detailed cable connectivity information from the sub-detectors, through intermediate patch panels, to the electronics racks. Cabling updates will be entered into the database using a barcode reader, and this data will then be exported to the online configuration/conditions databases, to allow channel mapping information to be automatically updated in the online and offline software. This facility will be particularly important during the commissioning phase, when many recabling operations can be expected.

4.3.4 Survey Database

A large amount of survey data is being accumulated during ATLAS integration and installation, ranging from measurements of the cavern floor displacements every few months through to the surveying of the internal geometry of assembled detector components and their final positioning in the cavern. This activity will continue during ATLAS operations, with e.g. the endcap calorimeter and muon wheel positions being resurveyed each time the detector is opened and closed. The data from such survey operations is typically processed by the CERN survey group, and made available to ATLAS in the form of reports and Excel spreadsheets. However, the data has potentially wide applicability for ATLAS reconstruction, e.g. for knowing the initial position of the muon chambers as a starting point for online and offline alignment. A survey database is therefore being designed, to centralize and make available data from survey group activities in a standardized way. This data will be stored on the Oracle physics database servers in the same way as other production and installation data, and will be made available to sub-detectors, e.g. through the Glance search interface or dedicated transfers to the conditions database.

4.4 Conditions and Configuration Data

4.4.1 Introduction

Conditions data refers to nearly all the non-event data produced during the operation of the ATLAS detector, together with that required to perform reconstruction and analysis. Conditions data varies with time, and is usually characterized by an 'interval of validity' (IOV), i.e. a period

of time for which it is valid, expressed as a range either of absolute times or run and event numbers. Conditions data includes data archived from the ATLAS detector control system (DCS), online book-keeping data, online and offline calibration and alignment data, and monitoring data characterising the performance of the detector and software during any particular period of time.

The basic function of the conditions database is to store the conditions data objects themselves, together with the associated IOVs. In some cases (e.g. DCS data), the data and IOV are tightly coupled, and stored together in the conditions database. In others (e.g. large calibration datasets), the conditions data can be created and validated without reference to any IOV, and only later assigned to one or more IOVs, when it is decided for which period(s) of time this calibration is valid. In this latter case, the conditions data may be stored independently of the relational database holding the IOVs, and the IOVs act as a cataloguing mechanism to index the conditions data objects by time, allowing the selection of subsets of calibration data corresponding to particular time periods. This view allows a distinction between the narrow IOV database, just holding the IOVs cataloguing the conditions data, and the wider conditions database, also including the data objects. When assigned to an IOV, the data objects are sometimes referred to as the payload of the IOV.

Conditions data is closely related to configuration data, needed to set up and run the ATLAS detector hardware and associated online and event selection software. Configuration data is characterized primarily by version or purpose (e.g. physics run, calibration or cosmic run) rather than interval of validity. However, once it is used for a particular data-taking run it becomes conditions data valid for a particular time, and it must be possible to recover the complete configuration in use at the time any particular event was taken. Particularly in cases when this data needs to be accessed from the offline reconstruction framework, this requirement suggests the use of the conditions database to store such configuration data. The ATLAS TDAQ project has developed a dedicated configuration database [4-2] based on the OKS [4-3] persistent in-memory object manager. This is widely used within TDAQ for configuring the online hardware and software systems and the data-flow system for the data transport and high-level trigger infrastructure. The TDAQ configuration database places great emphasis on good read performance, scalable to the thousands of processors in the online system. However, persistent storage is currently provided only by versioning the XML data files in a CVS repository, and there is no link to Athena. Some feasibility studies to improve archiving and implement Athena access are under way, and the possibility of replacing the XML files with a relational database backend is also being explored. However, the baseline plan is now to use the conditions database for sub-detector and trigger configuration data that is naturally stored in a relational database and needs to be accessed offline.

An ATLAS-wide conditions database was deployed for the 2004 combined test beam, and widely used for the storage of DCS data and calibration/alignment information. This database was based on a MySQL implementation of the conditions database interface developed by the RD45 project [4-4], with significant enhancements provided by ATLAS [4-5]. The database provided the association of conditions data objects with IOVs, with the conditions data itself being stored either within the database, or externally as simple structures mapped onto a separate MySQL database [4-6], or POOL objects stored in POOL ROOT files. Considerable experience was gained in both online and offline use of the database, and this has been very valuable input for the design and implementation of the new LCG conditions database product, COOL.

The final ATLAS conditions database will be based on COOL, making extensive use of all the features that this software offers. COOL is being deployed now for sub-detector commissioning in the ATLAS pit, and will also be heavily used in computing system commissioning late this

year and in early 2006. Various ATLAS-specific interfaces and utilities will be built on top of COOL, to support all the different types of conditions data in the experiment.

4.4.2 Conditions Database Architecture

Conditions data is usually characterized by an IOV and a data payload, with an optional version tag. The latter is only appropriate for some types of data, e.g. for identifying different sets of calibration data valid for the same IOV but corresponding to different calibration algorithms or reconstruction passes. The COOL conditions database supports both versioned and un-versioned data, and allows several possibilities for the storage of the data payload itself, either within the conditions database or externally in other databases or files. At the most basic level each set of conditions database objects corresponds to a relational table, with columns giving the interval of validity, an optional channel identifier (to allow several related but independent measurements to be stored together) and one or more data payload columns of simple types. The payload can instead be a reference to data stored elsewhere, e.g. a database table foreign key, a POOL token or an external file identifier. In these cases, the conditions database table is being used to catalogue or index data which is stored elsewhere, and may have an independent existence.

ATLAS plans to make use of all of these possibilities to store the various types of conditions data:

- DCS data will typically be stored directly in the conditions database tables; here the amount of payload data per IOV is usually small, and the data has no existence outside an IOV (i.e. a DCS measurement is always associated with a particular timestamp). This solution is also appropriate for other types of data where the payload is relatively small and the ability to make relational database queries on both the IOVs and the data payload is important. One typical example is monitoring summary data, where queries can be made to determine all runs where e.g. a particular part of the detector had high occupancy or was inactive.
- Payload data can be stored in the conditions database itself, but in separate auxiliary tables without an IOV structure, the IOV tables then holding foreign key references to row(s) in these payload tables. This solution is appropriate when the data payload is naturally represented in relational tables, but does not have a one-to-one correspondence with IOVs (either because one IOV points to several payload rows sharing a key, or the data payload can exist outside of the context of any IOV). Keeping such payload data directly in the conditions database has advantages both for ease of access and data distribution.
- Payload data can be stored elsewhere, either using POOL or in external files of arbitrary format. In the case of POOL, the data can be stored using the POOL storage service, using either streamed ROOT files, or the new POOL object-relational storage service. The latter is particularly interesting as it provides a way to map C++ objects into rows in relational database tables, allowing data to be manipulated both as C++ objects by programs using the POOL software, and by traditional relational database tools as might be used by online software or interactive users. In both cases, the conditions-database IOV tables store POOL object tokens (i.e. the string form of a reference to an object in a particular POOL streamed object file or relational database), allowing the POOL objects themselves to be retrieved from external storage. In the case of other file types, the conditions database simply stores the filename or other reference to the file, and external components are re-

sponsible for resolving the reference and providing the data to the application. Additional software will be required to manage and retrieve these files on demand. In the 2004 combined test beam, the CERN Castor system was used for this, although its not really suited to storing small files, and alternatives such as the SealZip package are being considered.

Sub-detector configuration data will typically be stored in the conditions database itself, either directly in the IOV tables or in other relational database structure. In the latter case, the data will be managed outside of the context of COOL, perhaps using the POOL RAL component to access the data in a database-backend independent way; this approach is currently being prototyped for the trigger configuration. COOL would then be used just to track which version of the configuration is used for which data-taking run. In all cases, updating the configuration will have to be done by creating a new version or new IOV, such that data which has already been used in previous data-taking runs is preserved in case it is needed offline. The version tagging features of COOL can also be used, to maintain several parallel versions of a configuration to be used for physics, cosmic or calibration runs. At the present time, sub-detectors (including the trigger system) are beginning to explore the possibilities of using the conditions database for configuration, and the first production use will occur in sub-detector commissioning beginning in spring/summer 2005. It is clear that understanding of the best way to use COOL for configuration data will grow as experience is gained.

ATLAS DCS data will come primarily from the distributed PVSS system controlling and monitoring the experiment and infrastructure. Around 100 PCs (running Microsoft Windows) are expected to act as sources of DCS data to be written into the COOL conditions database. These PCs will each run a dedicated application (known as the PVSS manager) responsible for subscribing to a set of PVSS datapoints and writing data updates to the conditions database. A first version of such an application was used extensively in the 2004 combined test beam to archive data to the MySQL-based conditions database, and much useful experience was gained. The total DCS data volume stored was O(10 GB), compared to around O(100 GB) per year being expected in the full ATLAS detector. An enhanced PVSS manager is now being developed, which will interface to COOL, and allow better configurability, both of which PVSS datapoints are stored in which conditions database table structures, and what filtering options (e.g. store on change, store at regular time intervals or on significant events such as start/end of run) are applied. Since around 100 separate PVSS manager applications are expected in the final system, efficient table structures (e.g. storing many independent but similar datapoints together in common tables), and buffered bulk updates (e.g. sending data as bulk inserts only every few minutes), will be very important. This enhanced manager will be used for the first time during ATLAS commissioning in 2005. Monitoring data will come as two basic types - summary tables consisting of numbers characterising parts of the system during e.g. one run, and histogram data. The former will be stored directly in the conditions database, with IOVs corresponding to the associated run, and the latter as histogram raw data e.g. in ROOT files. These files will be catalogued using the conditions database, with tables holding IOVs and references to the data files. As for conditions data POOL ROOT files, a system will be required for cataloguing and managing these external files. Little systematic work has been done on archiving monitoring histograms to date, but this is a priority area for the initial phase of sub-detector commissioning.

Online book-keeping data consists mainly of metadata on the runs and event data files produced during online data-taking. Some of this data will be stored in the conditions database, and some will be fed to the book-keeping database of the ATLAS offline production system. Several tools were used in the 2004 combined test beam to help with this. The online software 'conditions database interface' (CDI) allows datapoints of the online IS information system to be

monitored and logged to the conditions database [4-7], and this was used to store parameters of each data-taking run and the associated beam settings. This system is now being updated to interface to the COOL conditions database. Data was also transferred to the AMI offline bookkeeping database [4-8] to steer the subsequent reconstruction of combined-test-beam data and enable searches of the available datasets to be performed. This area will be revisited in the context of developments in the ATLAS production system and distributed data-management areas, and will have to be greatly expanded in functionality to deal with the multiple data sources in ATLAS, where event data will be written to many event filter sub-farm outputs (SFOs), with multiple streams (primary physics data, express physics stream, calibration and diagnostic streams) being produced in parallel. Close coordination with the production system managing the ATLAS Tier-0 reconstruction and calibration production will also be required.

Calibration and alignment data will utilize several of the possible data-payload storage options, with small data payloads being stored directly in the conditions database tables, and larger ones being written using POOL, either to streamed POOL ROOT files or with the POOL object-relational storage service. Storing data using POOL is particularly appropriate in offline calibration and alignment contexts, where the data is manipulated primarily as C++ objects, and may be produced in two steps; first as 'test' calibrations which are verified and perhaps iterated on, before being committed to the conditions database and assigned to intervals of validity. Several sub-detectors have explored POOL ROOT file-based calibration data in the 2004 combined test beam, and the LAr calorimeter has also used similar external storage in the legacy NOVA database system [4-5]. A similar scheme will be used for ATLAS commissioning and beyond, with the additional possibility of using POOL object-relational storage (expected to replace NOVA for LAr) to store calibration data in such a way that they can also be processed by relational database tools; this may be particularly useful for calibration constants that have to be shared between online and offline. The first version of POOL object-relational storage is available in POOL 2.0, and ATLAS integration is now being actively pursued, with a view to using it first for LAr calorimeter commissioning in summer 2005.

4.4.3 Conditions Database Online Model

The COOL conditions database software is written using the POOL RAL (Relational Access Layer) component, making the API and database schema independent of the underlying database backend (Oracle, MySQL and SQLite being currently supported). The large investment and experience of the CERN-IT database groups in Oracle, together with its proven ability to handle complex demanding applications, makes Oracle the natural choice for the conditions database implementation at CERN.

The CERN instance will have several simultaneous roles:

- Provide conditions data to the online system, record online book-keeping, DCS and monitoring information, and track online configuration and calibration data changes.
- Provide conditions data to the HLT system and Tier-0 processing farm.
- Support diagnostics, debugging and monitoring for online data-taking and prompt reconstruction.
- Act as the master repository for ATLAS conditions data, distributing data to the Tier-1 sites as needed, processing calibration updates received from the worldwide collaboration, and keeping backups and archives of old data as necessary.

- Provide conditions data for the CERN analysis facility, to support local physics analysis of CERN-based users.

It is clear that a single database instance will not serve all these roles, and a system with several replicas will be required. One possibility is to follow the example of the Tevatron experiments, with a master copy serving the online system, and read-only replicas for e.g. the Tier-0 and CERN analysis efforts. Such an approach was used successfully in the 2004 combined-test-beam MySQL-based conditions database, with offline reconstruction and analysis using a read-only replica of the main online database. However, in ATLAS itself, the online system will access only a fraction of the total conditions data, e.g. it will not require old data from previous years or data taking periods, or the most refined offline calibrations. Such considerations suggest that the online database server contain only the data required online, other data being kept on a separate 'offline' master server, with updates being fed from the online system as necessary. Prototyping and scale tests with the various Oracle replication and synchronization technologies are required before a final choice can be made, and such tests are planned during 2005. Another important issue is database server availability and fault tolerance. The conditions database will be essential for ATLAS data taking (particularly for online configuration of sub-detectors), so appropriate measures will be required (e.g. a fail-over server or standby replica) to ensure data is not lost due to database server problems. The physical location of the servers may also be important: in case of network problems or malicious hacker activity, ATLAS requires the ability to shutdown external network connectivity (including to the rest of the CERN site) and continue to take data for 24 hours. This suggests at least one standby server should be located at the ATLAS pit, but developments in network technology, firewalls and routing may mean that the server can still be located in the CERN computer centre and benefit from their database and server administration support. Again, more discussion and tests are required before a final decision can be made.

Database best practices suggest separating reading, writing and schema modification using distinct database roles. In the ATLAS conditions database, there will be many distinct writers (e.g. configuration, calibration and DCS for each sub-detector or subsystem), and these will have write privileges only for their own parts of the database to prevent interference. In contrast, a generic read role will probably be sufficient, since many readers (e.g. the HLT and Tier-0 farm) will need to access data from many sub-detectors at the same time.

Sub-detectors will also want to create new tables and modify their database schema, especially for ad-hoc table structures not created through the COOL API. However, this can have data-integrity implications on a production system, so it is anticipated that such operations will only be allowed freely on a development system, and new structures will only be introduced in production with the participation of the database coordinators. A review procedure will be required for all such changes, to ensure that the new structures use the database resources in an efficient manner and will not adversely impact existing users. This approach is supported by experience from the 2004 combined test beam, where free creation of new tables (particularly for DCS data) on the production server led to an unmanageable schema with 1000s of tables, and which proved very difficult to backup and restore.

Proper use of tablespaces and partitioning will also be vital to ensuring a scalable and maintainable conditions database. Table partitioning by IOV will be used to divide the data into time-based 'chunks', e.g. by year or smaller data-taking period, with older data being declared read only or removed from the online server. Efficient organization of data into tables, making full use of the 'channel ID' feature in COOL to store many related but distinct values into the same table, will also be important. Discussions with CERN-IT have emphasized the importance of good schema design to help ensure efficient use of database resources, and this will clearly be

vital for the ATLAS conditions database, with an expected data volume of several 100 GB of new data each year. Initial deployment of the online conditions database to support sub-detector commissioning at the ATLAS pit starts in spring 2005, with the first significant production use from sub-detectors expected in June. Although the data rates will initially be a small fraction of those expected in ATLAS, these rates will increase rapidly with the whole detector being commissioned in global cosmic runs at the end of 2006. Efforts will therefore be made to have as much as possible of the architecture of the final system in place early, with capacity being added to keep ahead of the increasing load from the ATLAS sub-detectors.

4.4.4 Supporting Tools

As well as the core conditions database software from the COOL project, several supporting applications will also be required. The online interfaces to the PVSS-based DCS system and the IS information service have already been mentioned above, but browsing, plotting and tag management tools will also be important. A PHP-based web browser was already developed for the MySQL-based conditions database used in the 2004 combined test beam, and this will be upgraded to interface to COOL and handle the various data payload types that are envisaged.

For DCS and other simple data types, a tool to dump conditions database folders to ROOT n-tuples is being developed. This could be run routinely on specified folders, for example at the end of each data-taking run, to provide n-tuples of recent DCS data that can be analysed without imposing the potentially chaotic load of interactive analysis sessions on the database server itself. Other more sophisticated interfaces to ROOT may be developed, depending on the progress of ROOT-relational database integration efforts.

The COOL conditions database includes a hierarchical tag model based on the HVS system, originally developed for the ATLAS detector description (see Section 4.4.7). This will be used to manage the calibration tags of individual folders, and to build up defined sets of tags for major reconstruction passes. The existing web-based HVS tagging interface will be adapted to facilitate this form of tag management.

4.4.5 Athena Access to Conditions Data

The Athena framework offers access to the conditions database through the interval-of-validity service (IOVSvc), which ensures that the correct calibration objects for the event being processed are always present in the transient detector store (TDS). An Athena job subscribes to a set of conditions database folders, and corresponding objects for the first analysed event are read in before user algorithm processing of this event begins. The service keeps track of the IOV of each object, and checks before processing each new event which objects, if any, are no longer valid and have to be replaced. In accordance with the Athena split of transient objects and their associated persistent representation, the actual reading of conditions objects is performed through conversion services, just as for event data objects. Writing conditions objects from Athena involves two steps: firstly recording the payload objects in the TDS and triggering the associated conversion services to write the objects on an output stream, and secondly registering the references to the written objects with associated intervals of validity in the conditions database. These two steps are currently being performed in the same job, but will eventually be doable in separate jobs (see below). The writing can be triggered from an algorithm execute method (i.e. during event processing), and work is in progress to extend this to allow also data to be written during the job finalization phase, once all events have been processed.

The scheme described above was used for the 2004 combined test beam, with payload object reading and writing through NOVA and POOL ROOT streamed file converters. A dedicated converter was also developed for conditions data payloads stored directly in the IOV tables, and used to access DCS and online book-keeping information directly from Athena. The software is currently being upgraded to interface to the COOL conditions database and the POOL object-relational storage service, to support access to conditions data that will begin to be stored in COOL as part of sub-detector commissioning in spring 2005. Mechanisms are being developed that allow individuals to produce calibration data sets and verify them before registering them with the main conditions database and distributing it for production use. This may be done by writing calibration data and reading them back directly without registration with an IOV, as mentioned above. It may also be useful to write calibration data and register IOVs in a temporary developer conditions database, and in a second step copy the data and IOVs to the main conditions database. Further developments will include support for pre-fetching conditions data in bulk operations, e.g. retrieving all the DCS data for a particular run in one operation, buffering it locally and then making particular values available in the TDS event-by-event as appropriate.

Use of Athena in the online HLT environment brings extra constraints from real-time operation, in particular that no conditions database access is allowed during the processing of a particular run. This implies that online processing (level-2 trigger and event filter) will use only constants that do not vary during a run, downloaded before event processing for that run begins. This also requires that critical data that may vary during the course of a run (e.g. DCS parameters or changes in readout conditions due e.g. to dead or noisy channels, the temporary loss of a data-taking partition or even a complete sub-detector) must be communicated to the HLT processors in another way, e.g. via information included in the event stream. However, such data may also be entered in the conditions database, for use in subsequent offline reconstruction processing.

4.4.6 Performance, Scalability and Distribution

Use of the conditions database online for sub-detector and HLT configuration presents considerable performance challenges. At run start, several 100 MB of constants will have to be downloaded to the various sub-detector controllers and processors in tens of seconds, often with multiple copies of the data being sent to several destinations. Similarly, the approximately 2000 HLT nodes will each require around 100 MB of configuration and calibration data, though in this case the data to be sent to each node will be similar. Offline reconstruction presents similar challenges, with Tier-0 and subsequent reconstruction involving 1000s of processors with the additional load from time-varying conditions data within one run.

It is clear that such parallel read performance is beyond the capacity of one database server, and that replication will have to be used to share the load amongst many slave servers. The task is eased by the fact that most writing will be done from a small number of well-defined sources (e.g. configuration updates from the online control system, calibration updates from sub-detector workstations or offline). The main issue is distributing these updates to all the replica servers. One interesting possibility comes from the Frontier project [4-9], developed to distribute data using a web-caching technology, where database queries are translated into http requests for web-page content, which can be cached using conventional web proxy server technology. This is particularly suitable for distributed read-only access, when updates can be forced by flushing the proxy caches, e.g. before a run start transition. Implementing the read part of the COOL API in terms of http requests would then provide a transparent way of implementing a scalable online database for configuration and conditions data.

Conditions data will also have to be distributed worldwide, for subsequent reconstruction passes, user analysis and sub-detector calibration tasks. The LCG 3D project [4-10] is prototyping the necessary techniques, based on conventional database replication, with an architecture of Oracle servers at Tier-0 (CERN) and Tier-1 centres, and MySQL-based replicas of subsets of the data at Tier-2 sites and beyond. The use of the RAL database backend-independent access library by COOL and other database applications will be particularly important here, to enable such cross-platform replication. It is also clear that conditions data updates (e.g. improved calibration constants) will be generated worldwide, and these will have to be brought back to the central CERN-based conditions database repository, for subsequent distribution to all sites that require them. Conditions data stored in files (e.g. POOL ROOT files and histogram data) will also have to be distributed, but, being file-based, this will be handled using the same data-management tools as for event data (see Section 4.6).

The performance, scalability and distribution requirements are extremely challenging, and a series of increasingly complex tests is planned to explore the limits of the planned solutions. Scalable performance has been a key design goal of the COOL API, and multiple-client read and write tests have already been performed in up to around 50 simultaneous sessions. Further tests are planned, in the context of commissioning the increasingly complex online computer systems at the ATLAS pit, and in the context of the calibration and alignment scalability tests during computing system commissioning in late 2005 and early 2006.

4.4.7 Detector Description

The general features of the ATLAS detector-description software are described in Section 3.5. The detector description is implemented as a series of modules, one per sub-detector, each of which reads the appropriate 'primary numbers' describing the detector geometry configuration from a database and builds the corresponding geometry classes.

The original version of the detector description used the NOVA MySQL-based database [4-5] to store the primary numbers, with different structures and table rows being accessed for different versions of the geometry, controlled by a number of ad-hoc mechanisms and switches. The primary numbers have recently been migrated to the CERN Oracle database and a system of version control has been introduced, based on the HVS hierarchical versioning system [4-11] developed specifically for this application. Each version of the geometry is tagged with an identifier (e.g. 'AtlasInitial', 'CTB04'), and this top-level tag identifies versions of data for each individual sub-detector and piece of sub-detector in a recursive tree-like structure, implemented via auxiliary tables in the relational database. The actual primary numbers are stored in payload data tables, and the tag finally identifies the row of each table that is appropriate for the geometry version being built. Both the hierarchical tagging mechanism and the data payload retrieval are implemented using the RAL database-independent access library, allowing the primary numbers to be stored either in Oracle or MySQL and facilitating worldwide replication and distributed access as discussed for conditions data above.

The detector-description database has been used as a test case for exploring database-resident data management and distribution procedures, in consultation with CERN IT application and database support experts. Sub-detector developers first enter new primary number data in a development database instance using a web-based tool, and can read from this database whilst debugging and validating their software. When a new release of the ATLAS detector description is being prepared, the HVS tag management tools are used to create a new top-level tag, and all the necessary information for this tag is copied from the development to the production data-

base, from where it is never deleted. This process is synchronized with ATLAS software releases, ensuring that production software always reads 'published' primary numbers from the production detector description database. This latter database can also be replicated from Oracle to MySQL, using tools based on the Octopus database replication software [4-12]. Tools have been developed to package and deploy such MySQL replica database servers e.g. for worldwide simulation and reconstruction in a Grid environment. All these steps have been successfully demonstrated and used for the Monte Carlo data productions performed for the June 2005 Rome physics workshop.

As with other types of configuration data, the detector description version associated with any set of event data must be stored. This is done using a 'RunInfo' object in the event stream, which contains the top-level tag used to produce the data, and will also be extended to include the conditions database tags used at each stage of the data processing. Eventually, this object itself may be stored in the conditions database, with an IOV corresponding to the run/event range in question.

4.5 Event Data

4.5.1 Introduction

The ATLAS event store is a multi-petabyte, globally distributed corpus of data describing real and simulated particle collisions and resulting interactions with the ATLAS detector. Event data are stored and managed at every stage of processing, ranging from readouts of the ATLAS detector written by the ATLAS Event Filter to highly distilled and derived data produced at advanced stages of analysis. The ATLAS Computing Model (Chapter 2) describes the principal processing stages (Raw, Event Summary Data, Analysis Object Data, and more) and event data flow. This information will not be repeated here.

This section describes the infrastructure used to write and read event data, including I/O infrastructure common to all data types, event-store technology choices, navigational infrastructure, principal event-store metadata components, and tools and services that support deployment and operation of a navigable, scalable, and maintainable event store.

4.5.2 Persistence Services

In the parlance of object-oriented programming, "persistence" refers to the capacity of an object to survive beyond the lifetime of the process that creates it. Persistence services provide the machinery to accomplish this, and encompass, for example, the means by which a data object's state may be saved in a file or database, and the corresponding means to restore the object to that state when it is needed as input to subsequent processing.

Athena inherits from Gaudi an approach to persistence that insulates physics code from specific persistence technologies. In this model, physics algorithms write and read data objects to and from an in-memory (transient) store that behaves as a blackboard for data sharing. In Athena, the blackboard infrastructure is known as StoreGate. Retrieval of an object from this transient store may in turn trigger data retrieval from files or databases, but this retrieval is intended to be transparent to physics users, as are any technology-specific operations involved in storage

and retrieval of that data. Control framework I/O infrastructure allows specification of which objects in the transient store are to be written to persistent storage, and when: for example at the end of an iteration of an event loop, or at the end of a job.

Providing persistence services to the ATLAS control framework requires more than delivering the capability to write objects and read them back again. The machinery by which the transient store supports object identification, aliases, base-class and derived-class retrieval, and inter-object references must also be persistified: one must not only save and restore (the states of) physics data objects, one must also save and restore the state of StoreGate. Many subtleties in the design and implementation of ATLAS event- store software arise from this fact.

Writers of event data specify which objects are to be persistified by means of an "ItemList," a term inherited from Gaudi. Entries in an ItemList must contain sufficient information to identify uniquely the objects to be written. In Athena, the corresponding information is typically the {type, key} values used for StoreGate retrieval, where "type" may be a class name, or, equivalently, a unique class id (CLID), and "key" is an optional (character) string in the C++ sense. More compact notation allows specifications like "ESD" or "AOD" as shorthand for specific ItemLists, the contents of which may have been predefined at an ATLAS-wide or physics-working-group level.

"OutStreams" control the writing of selected events. An OutStream is configured at runtime, at which point it is associated with a specific output technology and any corresponding technology settings (the filename, for example, when POOL ROOT is chosen as the output technology). The OutStream is also associated with a corresponding ItemList, and optional event selection criteria (typically a choice made by applying one or more physics "filtering" algorithms). A job may have many OutStreams, each with its own selection criteria and ItemList (and output technology). Thus a single job may write events of interest to a given physics working group to one OutStream and events of interest to another working group to another stream; moreover, the physics working groups may implement different policies regarding which event data objects are retained in their streams.

The ATLAS Computing Model proposes writing disjoint streams of AOD, but this is a deployment policy, not a design restriction: event-store software supports writing any event, at any stage of processing, to one or to several streams. The ATLAS Computing Model proposes also that "primary AOD" definitions (ItemLists) be the same for all streams, but this is again a matter of deployment policy—it is not a restriction imposed by event-store software. (Recall that primary AOD refers to data written at the Tier-0 centre as one of the products of initial reconstruction. AOD potentially produced later at Tier-1 centres by physics working groups may vary from one group to another.) By making these specific choices, though, the Computing Model ensures that there is a master AOD "dataset" in which each event appears exactly once, one that is capable of supporting selections for physics analyses that cross stream boundaries because AOD content does not vary by stream. In this model, streams play the role of a physical clustering optimization, placing like events ("like" by at least one criterion—that used for stream selection) proximately in persistent storage.

Within an OutStream, finer placement control will be possible by mid-2005. When an event is selected for writing to a particular stream, one may specify that calorimetry be written to one file and tracker data to another. While the current Computing Model does not propose to take advantage of this feature, its presence gives ATLAS the flexibility to optimize data placement in the event that certain kinds of data that must be retained for specific clients prove too unwieldy to deliver to every client of the data at that processing stage.

4.5.3 Storage Technology and Technology Selection

The ATLAS control framework and database architecture support runtime selection of storage technologies; indeed, the architecture supports simultaneous use of multiple technologies in a single job, e.g., POOL ROOT files for event data, a relational database for event-level metadata (tags), and still another relational technology for conditions or interval-of-validity data.

The ATLAS computing effort has over the years demonstrated the flexibility of its architecture and its ability to support a variety of persistence technologies, including an object database (Objectivity/DB), direct use of ROOT, and use of ROOT via the LHC Computing Grid (LCG) common project persistence infrastructure known as POOL. While heterogeneity of storage technologies in the ATLAS event store is therefore possible in principle, there are significant advantages to making specific technology choices for deployment at large scales, and this is what ATLAS has done.

ATLAS deploys two principal technologies for event data. All derived and simulated data use LCG POOL with ROOT files as the storage backend. Only data written by the Event Filter (and data associated with Event Filter simulations) are different: raw event data are written into "bytestream" files in a format matched to ATLAS detector readout.

The LCG POOL project provides not only a persistence technology but also a capability to support runtime storage technology selection. POOL's newly introduced Relational Access Layer and object-relational mapping layers make it possible in principle to decide at runtime whether to write data to a POOL ROOT file or to a relational database. ATLAS retains its own technology direction layer, which predates the POOL project; thus, there are multiple loci in ATLAS software at which one might plug in new or alternative storage technologies should the need arise.

4.5.4 Persistence of the Event Data Model

A requirement of any event-store persistence-technology candidate is the ability to support the ATLAS event model. This may not necessarily require the ability to persistify every C++ language construct, but any limitations on what can and cannot be persistified must not unduly constrain the event data model. While the event model has not yet been finalized in every detail, an important event-store milestone was met in 2004: "POOL demonstrably capable of supporting the ATLAS event model," confirmed by a series of tests using ATLAS Data Challenge 2 data and the then-current event data model. Minor compromises were made and issues associated with persistence of certain CLHEP classes were uncovered, so work continues, but no major impediments to use of POOL as the primary event-store technology were found.

4.5.5 Automatic Persistence

While software is in development, it is extremely convenient to be able to say "store my object" without concerning oneself with its persistent representation. In the Gaudi/Athena architecture, runtime specification of an output technology triggers loading of a technology-specific "conversion service", which provides the "converters" to map from transient-data object structures to possibly technology-specific persistent layouts and back again. When these converters can be generated automatically, it is a boon to developers. The ROOT and LCG SEAL and POOL efforts have delivered the means to accomplish this for a wide range of data types (with perhaps the aid of a selection file to indicate which data members need not be persistified); in-

deed, automatic converter generation is standard practice for today's ATLAS event model. When such an automatic mapping is impossible or inappropriate (e.g. inefficient in terms of space usage), a capability to invoke hand-coded converters is also provided.

An artifact of a pure persistence model is that the reader's view is the writer's view: one takes a snapshot of data objects in the transient store after a processing stage, and later readers may retrieve some or all of these objects. Often this is precisely what is wanted; the first algorithm of the next job picks up where the last algorithm of the previous job left off. It is not a view typically taken in databases, though, wherein clients extract the view they desire in a manner that may be conceptually independent of any particular data-writer's view.

ATLAS event-store persistence infrastructure has been designed to accommodate delivery of a client view (transient objects) different than that recorded persistently by providing a means to distinguish transient from persistent type identification, and machinery in both ATLAS-specific and POOL layers to handle cross-type mapping. (In principle, the persistent view need not be the writer's view either: the same machinery can support transient-to-persistent type remapping on output.) When ATLAS data taking begins, it is expected that the pure persistence approach (writer's view and reader's view are essentially the same, and persistence captures the writer's view) will dominate, but the more general capability incorporated into the event-store technology design may grow in importance as schema and even language choices evolve.

4.5.6 Navigational Infrastructure

In addition to support for persistence of inter-object references in the transient event data model, the ATLAS event store also retains navigational information to allow access to upstream data.

The model is this: when an event is written via an OutStream, each event data object in the associated ItemList is written. In addition, a master object (a DataHeader) is written, which tracks where each individual physics object has been written, along with its StoreGate identity (type and key), and any other information needed to restore the state of StoreGate. This DataHeader serves implicitly as the entry point to any event: when one retains a reference to a persistent event, it is in fact a reference to this DataHeader.

A DataHeader also retains references to DataHeaders from upstream processing stages. This allows back navigation to upstream data, so it is unnecessary to rewrite, say, Monte Carlo truth data in one's AOD, though one may certainly do so as a performance optimization. Back navigation happens by recursive delegation: a reader of AOD who asks for calorimeter cells (by type and key, in the StoreGate fashion) will trigger a retrieval request from the input (AOD) event, which will not find anything with that type and key attached to its DataHeader. It will then delegate the request to its parent DataHeader, and so on. Back navigation may optionally be turned off to avoid undesired traversal of multiple files, which may or may not be locally available. More sophisticated back navigation control is under development.

4.5.7 Schema Evolution

The ATLAS event data model will almost certainly change over time, and event-store planning must anticipate this. ATLAS must retain the ability to read both old and new data after such a change, regulate the introduction of such changes, minimize the need to run massive data con-

version jobs when such changes are introduced, and maintain the machinery to support such data conversions when they are unavoidable. In database literature, such changes to the layout of persistent data structures are known as schema evolution.

ATLAS has chosen a technology for its event store that already provides the capability to accommodate minor schema changes (addition or deletion of data members, for example), and these capabilities are improving. For more substantial changes to the event data model, the architecture provides at least two loci for introduction of code to detect and compensate for changes in persistent-data format. This work can be done at the "converter" level in Gaudi/Athena persistence layers, or, at a lower level, via "custom streamers" in POOL. Tools to detect schema changes from one release to another have been delivered; these provide an aid to support policy-level control of changes to the content definition of endorsed event-store data products at various processing stages.

4.5.8 Event-level Metadata

In a multi-petabyte event store, a means to decide which events are of interest to a specific analysis without first retrieving and opening the files that contain them, and to efficiently retrieve exactly the events of interest and no others, is an important capability. For standard analyses, standard samples will be built, according to criteria provided by physics working groups. Streams written as a product of primary reconstruction at the Tier-0 centre provide one means of building these samples. For analyses that require samples that cross stream boundaries, or for finer subsamples within a standard dataset, event-level metadata (tag databases) provide a means to support event selection.

When an event is written, metadata about that event (event-level metadata) may be exported, along with a reference to the event itself. Quantities chosen for export are those that are likely to be used later for event sample selection, and are defined by physics and physics-analysis-tools working-groups. These event tags may be written to a POOL ROOT file, or to a relational database.

RegistrationStreams provide the control framework machinery to accomplish this export of event-level metadata, and POOL explicit collections provide the implementation An event of interest to multiple analyses may be written exactly once, but the corresponding event tag and pointer to the event itself may be recorded in multiple collections; alternatively, the tag may be written to a single collection, with subsequent queries used to build multiple collections corresponding to multiple, specific analysis selections. Event-store back navigation machinery provide the means by which a collection built, for example, after AOD has been written, may nonetheless support selections of data from upstream processing stages.

4.5.9 Event Selection and Input Specification

ATLAS event-store infrastructure provides many means to specify input to an event-processing job. The means most frequently utilized today is the name of a single file, but the infrastructure supports also: lists of files (logical or physical), dataset names that map to lists of files, lists of specific events (via event collections/tag databases), specific event sets with filter criteria (query predicates applied to event collections), and heterogeneous combinations of all of these.

Only the primary input events need be specified to the Athena EventSelector. One specifies a file containing AOD, for example, as input, and back-navigation machinery (in the presence of an appropriate file catalogue) will find and open ESD if necessary.

4.5.10 Event Store Services

A variety of tools and services support identification, selection, navigation, and extraction of data samples of interest in the event store. Metadata associated with datasets (both file sets and event collections) support queries to identify which datasets are of interest. Middleware typically maps such selections into lists of logical files (or lists of file sets used as units of data distribution), which can then be located by other middleware services, including one or more replica catalogues. File management and cataloguing is discussed in Section 4.6.

Typical output of an event-level metadata query is a list of references to events that satisfy the selection predicate. Such output suffices for jobs that run, for example, at a Tier-1 centre, where all AOD reside on disk, but may not suffice for laptop-level analysis. For such purposes an associated utility to extract replicas of exactly the selected events and no others into one's personal files is also provided. A related utility that builds the list of logical files that contain the selected events has also been delivered.

A variety of diagnostic utilities—to determine the contents of an event data file, to verify that all data objects in the file are retrievable, and so on—have been deployed, and the list is growing.

4.5.11 Summary

One guiding principle of event-store architecture has been to design for generality, and to support specific deployment choices and constraints by means of (enforceable) policy. Thus the design supports multiple persistence technologies, but ATLAS will deploy a specific one (ROOT-file-based LCG POOL); the design supports possibly overlapping streams, while the computing-model policy may keep streams disjoint; the design supports content definitions that may vary from stream to stream, while the Computing Model may call for keeping definitions identical; the design supports arbitrary transient/persistent type remapping but deployment at turn-on may not exploit this. A second principle is that the user view of event data storage and access is the same for laptop users as for distributed Grid-production jobs—only the specific data sources and scale change.

4.6 Distributed Data Management

4.6.1 Introduction

At the time of writing ATLAS has recently begun development of the final distributed data management (DDM) system for startup, DonQuijote 2 (DQ2). The scope of the system encompasses the management of file-based data of all types (event data, conditions data, user-defined file sets containing files of any type). The requirements, design and development of the system draw heavily on the 2004 Data Challenge 2 experience gained with the ATLAS-developed DonQuijote distributed data manager (DQ) and the Grid middleware components underlying it.

The other principal input is the ATLAS Computing Model, which provides a broad view of the environment and parameters the DDM system must support. The overall objective for DQ2 is to put in place as quickly as possible a highly capable and scalable system that can go into production as soon as possible, and which can be incrementally refined through real usage to be ready for computing system commissioning and data taking. During 2005 it will support steady-state production, increasing analysis activity, the onset of commissioning, and preparatory testing on the Service Challenge 3 (SC3) service in advance of the computing-system-commissioning scalability tests (DC3) coming in early 2006. A high-level timeline for DQ2 development can be found in Section 4.6.8.

DQ2 carries over the basic DQ approach of layering a stable and uniform experiment-specific system over a foundation of third-party software (Grid middleware, common projects) while making many changes at all levels from design precepts to implementation. The approach is well suited to supporting our multiplicity of Grids; accommodating a time-varying mix of middleware as components mature; and integrating pragmatic in-house developments that provide experiment specificity and fill gaps in the available middleware. The present DQ consists of three main components: a uniform replica catalogue interface fronting Grid-specific replica-catalogue implementations; a reliable file-transfer service utilizing a MySQL database to manage transfers using gridftp and SRM; and Python client tools for file lookup and replication. Further information on DQ can be found at ref.[4-13].

Continuity in meeting data management needs during DQ2 development is being provided by sustaining support and essential development of DQ until a transition to the fully-deployed production DQ2 is completed in late 2005.

The next subsection summarizes the DC2 experiences that to a great extent underly the design and implementation decisions taken with DQ2. The rest of the section focuses on DQ2: design precepts and requirements; scenarios and use cases; system architecture; implementation and evolution; and development, deployment and operations planning and organization. The DQ2 system design is based on an analysis (following the Computing Model) of data management in ATLAS including managed production, global data discovery, managed distribution operations, and end-user analysis support including ‘off-Grid’ usages.

4.6.2 DC2 Experience

During DC2 a number of data-management shortcomings were found, many originating in problems with or shortcomings of the Grid middleware. Grid replica catalogues were a principal source of problems. The catalogues from the middleware providers of the regional Grids were adopted and used "as-is" by ATLAS. The granularity at which the catalogues operated (individual files) and the functionality provided (simple inserts, updates and queries) proved to be insufficient. Bulk operations were needed for realistic use cases but were not supported. Servers were either not designed to be deployed on several geographically distributed locations or failed to work reliably when these setups were exercised. Some problems were partly solved during DC2, but experience generally showed that much of the middleware was put into production without proper testing cycles. Addition and updates of replica-catalogue entries was not a problem for DC2, mostly due to the fact that these operations were issued by single (or very few) jobs at a time. Problems while adding or updating entries were encountered when the replica-catalogue servers were unavailable, but this was mostly solved during DC2 and currently the servers very rarely fail. Querying was and still is a critical bottleneck. Efficient queries cannot be performed at all. Relying on application-specific metadata associated with each file

proved to be very slow and its usage for querying was mostly dropped during DC2. Indexes on key elements (such as storage locations) are still missing. Collections/grouping of files is not supported, which means scoping of queries is not possible (full replica catalogue queries are in practice quite common).

Besides replica catalogues, the lack of a robust and unique information system also proved to be a problem. There is no automatically managed, reliable listing of ATLAS storage resources indicating the capabilities, reliability and QoS of each resource, as negotiated by ATLAS. The data transfer components provided by the Grid services didn't provide a minimal level of managed functionality for most of DC2. SRM is still not broadly deployed, so ATLAS itself has the task of debugging GridFTP server errors and trying to understand the cause of site problems. This infrastructure should be managed by the Grid middleware itself and meaningful errors should be reported back.

Prior to the start of DC2, DonQuijote was developed as a small tool to locate files and issue file movement requests between Grid flavours using Grid middleware. It was expanded in the light of experience to compensate for missing (or not working) functionality. DonQuijote eventually became a single point of failure since all production jobs depended on its functionality. The DonQuijote architecture was not originally designed to scale to handle production requests, but this was mostly solved during DC2. Nevertheless, DonQuijote was only partially successful in its goals. The file validation performed by DonQuijote at file registration proved insufficient, because in real operation entries that once were valid become corrupted later. This exposed the need for a continuous validation process to routinely check catalogues and sites for incorrect entries. Implementing this was not practical due to poor performance of the catalogues, and also due to the fact that DonQuijote (and the replica catalogues) do not understand the 'dataset' concept, so organizing validation in a scalable way above the file level was not possible. Also, the lack of validation features on GridFTP servers (or alternatively the lack of a managed, reliable file transfer component) was a further problem. Site disks failed and inconsistent replica-catalogue entries had to be manually fixed by ATLAS. Problems were also found in two areas important to end users: delivering reliable end-user client tools for accessing data, and reliable file movement. Both are still much too dependent on low-level Grid components which are not yet stable, GridFTP front-end servers to mass storage systems and slow queries on replica catalogues being the main bottlenecks.

At the operations level, there was a shortage of manpower especially in the areas of networks and data transfer (to interact with individual sites) and monitoring. There was insufficient communication and coordination between sites and ATLAS production managers in ensuring that sites were provided with needed input data by the time processing was scheduled to start, and that production outputs were promptly made available.

These experiences have provided important lessons and guidance for the design of the next-generation system described below.

4.6.3 Precepts and Requirements

Here follow the principal design precepts and requirements driving the design and implementation of DQ2.

- **System architecture should support 'at-scale' usage.** The DQ2 architecture should support the scale and functionality required for ATLAS running; i.e. it should have the capability to support the at-scale ATLAS Computing Model. The system design should

anticipate progressive refinement as we accrue experience and as more capable and robust implementation software becomes available. Neither the time nor the manpower will be available to go through a later redesign of the system architecture to meet final ATLAS requirements.

- **System architecture should be realizable in an implementation now.** The DQ2 architecture must be implementable now using existing, proven software and tools. We must be able to build a capable system now, following the architecture, that we can use, learn from and refine. Time and manpower (and good practice) preclude divorcing the final system from our near-term production system; they need to be one and the same.
- **The initial implementation should use the most robust, functional and scalable solutions available (or readily implementable) today.** The initial implementation will provide our production data-management system for the near term. It must be as robust, functional and scalable as possible, both to meet production demands and to exercise the architecture at the largest achievable scale. Initial implementation choices should accordingly be driven by choosing the approaches, tools and technologies which deliver the best capability today, within the limits of what is realizable with the available manpower.
- **The design should support progressive introduction of new middleware.** The initial system will not use the full suite of middleware we expect to use in the future. Some components are at this point unavailable, immature or unproven. By supporting and using agreed standard component definitions and interfaces, the design should support the progressive introduction of new middleware components over time as they mature.
- **Maximizing performance should be a design and implementation priority.** Performance and scalability should be a principal consideration from the earliest stages of design through implementation.
- **Quantitative benchmarks and metrics should govern the introduction of new components.** Quantitative measures and requirements should be established, based on experience with the initial system, on which to base decisions on the introduction of new middleware and other components.
- **Multiple Grids must be supported.** The present multiplicity of Grids must be supported by the system, hiding their details from the naive user while providing the flexibility to work with the different Grids, including regional tools such as Grid or SE-specific replica catalogues.
- **The system should coherently cover the full scope of use cases from on- and off-Grid analysis users through large scale production.** The design and implementation must support end-user analysis usages, including off-Grid endpoints (e.g. laptops), as well as production usages. The priority given to supporting end-user usage should be at least as high as that for production usage.
- **Datasets should be the principal means of file-based data organization and lookup.** In order to achieve adequate performance and scalability in lookup and access of data files, and in order to provide the data aggregations required for efficient data handling and analysis in support of the ATLAS Computing Model, the DDM system should work wherever possible with file aggregations (datasets) rather than files.
- **The system must provide data management tools to end users at the file level as well as the dataset level.** While most end-user data access will be done at a higher (dataset) level, working with files (and collections of files) is important for end users today and will remain so in the future in some contexts. Consequently, while the system should be optimized for dataset-level access, it must also support file-level access.

- **'Shared' as well as 'official' data must be supported.** In addition to supporting archiving, registration, discovery, replication and access for 'official' ATLAS data (raw data, data from managed production operations from simulation through analysis, etc.), the system should support these functions also for 'unofficial' data shared among individuals or within groups (analysis groups, calibration teams, etc.). Efficient discovery and access throughout ATLAS is required for all official and shared data.
- **Read control must be available and write control must be in place.** The system must be capable of limiting read access to ATLAS VO members, if and when ATLAS policy dictates. Until such a policy is in place, the system should not require user authentication for read access. For all write and delete operations the user must be authenticated and must hold role-based rights permitting these operations.
- **GUIDs and logical filenames (LFNs) will be used for logical file identification.** GUID-based lookup of logical files (we use the POOL-defined GUID for POOL files) will be the most common access mode, but LFN-based lookup will also be used, e.g. for end-user access at the file level, for browsing and for debugging. Consequently, the system should support both GUID and LFN-based lookup.
- **LFN renaming and aliasing are specifically excluded.** A consequence of supporting both GUID and LFN-based lookup with adequate performance is redundant recording of GUID-LFN associations in many places (see the catalogue-content descriptions that follow). This has severe implications for the cost and practicality of supporting either the renaming of LFNs or the aliasing of multiple LFNs to one logical file (GUID). Consequently, in the absence of important use cases for these functions, LFN renaming and aliasing will not be supported.
- **Files are mutable during preparation, immutable after publication.** Files may first be registered in the DDM system during their generation/preparation phase at which time they are mutable – writing is in progress or appends are possible. However, files in the mutable state are not available for replication by the system; replication is only allowed once files are complete and marked permanently immutable. Thus no support is required for file versioning. File-level metadata must include mutability state, used by the replication services to disallow replication of mutable files.

4.6.4 Scenarios and Use Cases

The scenarios and use cases following from the ATLAS Computing Model that drive the DDM system design are described here.

4.6.4.1 Data Acquisition, Managed Production

Raw data flowing into the Tier-0, managed production of ESD and AOD, and simulation production will all yield datasets (file aggregations) of an operationally convenient granularity for cataloguing and distribution. Raw data files for a particular run may constitute one dataset, for example, with associated ESD and AOD another, and all of them aggregated for distribution to a Tier-1 centre. Production operations will rely on input datasets distributed in advance and available at time of processing at the production site.

4.6.4.2 Data Aggregation and Splitting

Data aggregation may occur at multiple levels for multiple purposes. In the example above, a RAW+ESD+AOD aggregation of all produced data is appropriate for defining the data units to be replicated from Tier-0 to a Tier-1. Replication of analysis data to a Tier-2 would be on the basis of a different aggregation, e.g. all AODs from streams used in a Higgs analysis. Data aggregrated for replication or other purposes may later have to be split into subsets; for example, a replicated block of data may be subdivided at the destination for further distribution of subsets to processing farms. The system should simultaneously support such multiple, overlapping, layered aggregations (ie dataset aggregations, which may be hierarchical, with a given dataset able to appear in multiple aggregations/hierarchies).

4.6.4.3 Group-level Data Production (analysis group production, calibration teams)

Analysis groups and calibration teams will also conduct managed production-type operations, with workflows similar to (but more frequently iterated than) managed production. Data distribution patterns will also be similar.

4.6.4.4 Managed Data Distribution

Datasets created in DAQ, production, calibration, and analysis operations need to be replicated to interested recipients in an automated manner. Managed distribution operations need mechanisms to establish 'subscribers' to particular data sets and have automated replication proceed accordingly. The system should support automated replication not only of static, predefined sets of data but dynamically changing sets (e.g. 'replicate all new Tier-0 ESD datasets to institute X'). The system should allow for use of replicated data at the destination prior to completion of the replication (e.g. commence processing of an imported ESD dataset when the first constituent file arrives, not the last).

4.6.4.5 File Migration

Automated mechanisms for reliably migrating files produced at a source to an archival destination, with the files catalogued for accessibility later, must be provided. For example, the migration of data files as they are produced online to Tier-0 mass store.

4.6.4.6 Data Discovery and Access

Data discovery at the dataset level and the file level must be fast and efficient ATLAS-wide: from anywhere in ATLAS it must be possible to discover datasets and files residing anywhere in ATLAS. Similarly ATLAS-wide replication must be supported down to the file level. From a regional Grid production system needing to locate and replicate an input dataset to an individual physicist needing a particular raw data file to diagnose a detector problem, it must be possible for anyone anywhere to locate efficiently and – subject to cost controls – replicate data residing anywhere.

4.6.4.7 Physics Analysis

To support physics analysis, essentially the full capability of the system (definition and cataloguing of datasets, data discovery, data replication and access) needs to be made available at the level of the individual user – of which there will be many hundreds at any one time – with appropriate controls on access, individual resource usage, and collective end-user resource usage relative to production usages.

4.6.4.8 Regional Semi-Autonomous Grids

Regional Grids will be engaged in both ATLAS-level operations (the 'ATLAS share' of regional resources) and regional operations (autonomous activity driven by the interests of the region). The DDM system should smoothly integrate ATLAS-level regional operations into the overall ATLAS DDM. The DDM system should also be usable (at the discretion of the region) for autonomous regional operations.

4.6.5 System Architecture

The system architecture arrived at to address the requirements and usage scenarios described and support the ATLAS Computing Model is based on hierarchical organization and cataloguing of file-based data, and the use of these hierarchies (there is not one unique hierarchy) as the basis for data aggregation, distribution, discovery and access. Files are aggregated into datasets, datasets are aggregated by dataset containers, and dataset containers can be organized in a hierarchy to express flexibly layered levels of containment and aggregation. None of the assignments (file to dataset, dataset to container, container to hierarchy) need be unique; different aggregations will be appropriate in different contexts, and the architecture has the flexibility to express whatever aggregations or 'views' of the data are required.

The data aggregations used as the units of data distribution around the collaboration provide a natural unit for looking up data locations, and the architecture is based on these (data blocks). Global data lookup at the dataset level rather than the file level affords better scalability in two respects: the entity count is down by orders of magnitude (we expect far fewer datasets in the system than files), and any dataset attribute may be used as the basis for partitioning catalogues into multiple instances (e.g. a dataset location or content lookup could first resolve that the dataset belongs to running period 2009a, and accordingly route the lookup to a catalogue instance for that running period). P2P technologies and searching algorithms may prove useful for steering queries to catalogues when we have multiple instances of them; this will be explored in the implementation.

Datasets and their containers that are used for distribution and data-location discovery are required to be immutable once their content is defined, such that the content of these data blocks is uniquely defined and stable. This allows the dissemination of this information around the collaboration for rapid, scalable lookup of data location without introducing a complex catalogue-consistency problem.

The principal components and features of the architecture are described in the following sections. Figure 4-1 shows the major catalogues in the system and the interactions with DQ2 servic-

es and user applications (production operations, end users). Initial implementation choices for the current prototype (discussed in Section 4.6.6) are also shown.

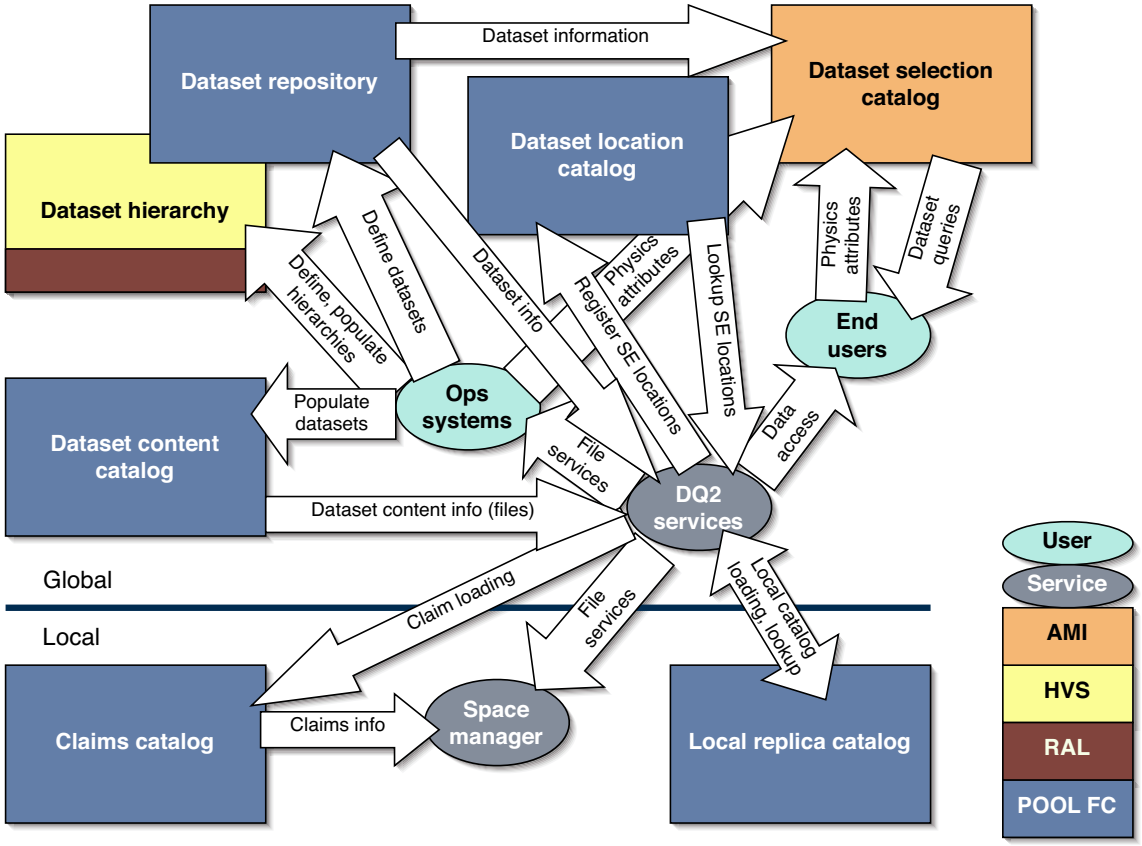

Figure 4-1 Architectural overview of the DQ2 distributed data management system

4.6.5.1 System Components and Terminology

4.6.5.1.1 Dataset

For the purposes of the DDM system a **dataset** refers to an aggregation of files plus associated metadata descriptive of the dataset itself and its constituent files. Datasets typically contain event data sharing some association (temporal, physics selection, processing stage, etc.), but can contain any sort of file-based data. A dataset is identified by a unique **dataset ID (DUID).** All files managed by the DDM system are constituents of datasets. A file can be a constituent of multiple datasets. Constituent files are identified by logical identifiers (LFN, GUID). Datasets hold a **mutability state** which can be unlocked, locked or frozen (permanently locked). Datasets that are not frozen can be versioned, to support discrete changes in their content tagged by version IDs. Different versions of a dataset share the same DUID; the composition of a DUID with version ID defines a version-unique identifier, **VUID**. The content of mutable datasets can be changed in any manner (additions or deletes).

4.6.5.1.2 Container dataset

Datasets can themselves be aggregated into **container datasets** consisting of other datasets (including other container datasets), plus metadata associated with the container dataset. Container datasets cannot directly contain files; only other datasets. Container datasets thus implement a hierarchy (tree) of datasets. Container datasets also hold a mutability state and (if not frozen) a version.

4.6.5.1.3 Dataset hierarchy

The **dataset hierarchy** implements hierarchical organization of datasets via container datasets. A dataset hierarchy is not intrinsic to the datasets involved; the hierarchy is an externally imposed organization of pre-existing datasets. A dataset can appear in any number of hierarchies. The dataset hierarchy supports this flexible organization of hierarchical dataset trees of arbitrary depth, with support for versioning at the nodes of the hierarchy such that the dataset content of a container dataset can be changed and versioned.

4.6.5.1.4 Dataset repository

A **dataset repository** is a catalogue of datasets. Each dataset is represented by one entry in the catalogue, identified by a unique ID (DSID) and a name string (also unique). The repository catalogues the DDM system metadata for datasets, and has the ability to catalogue any content-specific attributes of the dataset that it is found to be useful to record here (it is unspecified at this point what, if any, metadata not strictly needed by the DDM system should be recorded in the dataset repository). All dataset versions are catalogued in the repository. The combination of DSID and version ID uniquely identifies a dataset version (DSID is the same across dataset versions).

The dataset repository serves as the principal catalogue and lookup source for datasets defined and made available by the DDM system. It catalogues container datasets as well as file datasets. The repository is not the means by which physicists perform queries to select datasets of physics interest. Rather, the dataset repository must support the efficient export of dataset information to physics-selection catalogues and tools such as the dataset selection catalogue. The dataset repository defines the dataset in terms of its data content; the dataset selection catalogue adds descriptive queryable information such as physics content, data quality, luminosity etc. (information which may change over time).

There will not be one monolithic dataset repository. The repository will be partitioned into multiple instances on the basis of function (different instances possibly having different attribute sets), scope (e.g. distinct repositories for local and global datasets), and for scalability purposes (e.g. by running period).

4.6.5.1.5 Dataset selection catalogue

A **dataset selection catalogue** is the catalogue by which physics selections are made to identify datasets of interest. It is not a part of the DDM system, but receives information from the DDM system (the dataset repository) on file-based datasets and associated metadata. Other data sources add physics metadata to the dataset attributes recorded in the dataset selection catalogue to enable physics selections.

4.6.5.1.6 Dataset content catalogue

A **dataset content catalogue** records the logical file constituents of datasets. These catalogues record all (logical) files managed by the DDM, with global scope, so scalability is a major issue. Organization of the files in terms of datasets is the means to achieving scalability. There will not be one monolithic dataset content catalogue. An ensemble of content catalogues partitioned according to dataset metadata will share the cataloguing load. Dataset content lookup will proceed by 1) dataset lookup in the repository, retrieving metadata; 2) identification of the content catalogue, either rule-based using dataset metadata or via explicit recording of the content catalogue in the repository; 3) content lookup in the content catalogue. Dataset-dependent determination of the appropriate content catalogue requires this one pass through the repository, but the benefit afforded is flexible scalability in content lookup through content-catalogue partitioning.

Dataset content is made available in global scope in order to support file-level lookup with reasonable efficiency. File lookup will generally proceed from a context in which the file is known to reside in one of a known ensemble of datasets (from the processing history), enabling quick determination of the containing dataset via the content catalogues and thereby quick resolution of site-level file location via dataset location lookup mechanisms.

Physical file locations at a site are available only at the site (SE) level (the level at which this information is managed and relevant), via the local replica catalogue.

4.6.5.1.7 Data block

A **data block** is a frozen (permanently immutable) aggregation of files for the purposes of distributing data and thereafter finding it again. A data block is an instance of a dataset or container dataset. That is, it can be directly a collection of files, or a hierarchical collection of datasets. Constituent datasets within a data block are themselves data blocks (i.e. must be frozen).

Data blocks are the unit of data replication and data location lookup used by the DDM system. A data block must be wholly resident at a given location (site or SE), modulo temporary incompleteness during block creation or transfer, in order for it to be registered in the system as present at that location.

When a data block is replicated to a given location, the block's presence at that location is catalogued (via the data location service below). This catalogue provides the mechanism for discovering data location. This usage is why data blocks are required to be frozen: if their content were allowed to change, ensuring the correctness and consistency of data block location catalogues to support data lookup would be much more difficult.

Data block constituents of a container block can themselves be registered in the location catalogue, thus supporting different granularities of data distribution appropriate in different contexts (e.g. dissemination of RAW+ESD+AOD from T0 to T1 vs. a T2 picking up AODs only from a T1).

4.6.5.1.8 Data location service and catalogue

The **data location service** provides lookup of the site locations at which copies of data blocks can be found. It uses a **dataset location catalogue** to record data block locations. When a site obtains a replica of a data block, it registers the presence of the block at the site via the data loca-

tion service. A container block can be registered only if all constituent blocks are available (and any constituent blocks which might be used on their own should also be registered).

4.6.5.1.9 Data subscription service and subscription datasets

The **data subscription service** enables users and sites to obtain data updates in an automated way via 'subscriptions' to mutable datasets, **subscription datasets**. Subscription datasets may contain either files or other datasets (container datasets). Subscription datasets are versioned, so their content can change discretely by version over time. When a dataset is updated, it is unlocked, its content is updated (with insertions and/or deletions), the version tag is updated, and it is locked again. The data subscription service can then propagate the new version to subscribers. A 'continuous update' mode will also be supported in which dissemination of updates is triggered not by appearance of a new version but by any change in the subscription dataset's content (first and second use cases below). The subscription system is implemented at the ATLAS application level as a means of automating and managing data movement, and makes no special demands on the underlying middleware and facilities services used to move data.

Subscriptions are inserted in the dataset location catalogue.

Subscriptions can serve a number of purposes, e.g.

- managed distribution of data blocks. A container dataset might contain all the raw and first-pass ESD data blocks destined for a particular Tier-1 facility. The facility subscribes to this dataset, and Tier-0 production operations populates it with new raw+ESD data blocks as they appear, triggering automatic replication of the blocks to the Tier-1.
- automated movement of raw data at the file level from event filter to Tier-0
- automated dissemination of physics-working-group datasets. A group might disseminate AODs from managed AOD production operations to end-user or institutional subscribers.
- distribution of data files associated with software releases. Subscription datasets containing data files, test files, etc. that need to be propagated with the release could be a simple means of disseminating this material.

4.6.5.1.10 Local replica catalogue

The globally scoped components of the DDM system record file information only at the logical level (GUID and LFN). The physical location of a file on a storage element is information specific to an SE, and so is recorded (only) at SE scope, in a **local replica catalogue** at the SE. The 'local' scope of a replica catalogue is subject to ATLAS policy, which may be Grid-dependent; the scope may be one SE (most typical), or a set of SEs/sites within a Grid (e.g. to support a set of small sites via one catalogue), or even an entire Grid (the approach used in the present system, but not scalable and therefore not likely to be carried over to the new system). In addition to scalability, delegation of physical replica cataloguing to sites affords them the ability to flexibly and autonomously manage data location on their SEs. The limited scope and therefore scale of the local replica catalogue makes the logical-to-physical deference a minor overhead in navigating to physical files.

4.6.5.1.11 Space management service and claims catalogue

Another component of local rather than global scope is the **space management service**. Users and production jobs can pin data on disk by registering a 'claim' on a disk-resident dataset or file with specified lifetime. Claims are recorded in a **claims catalogue**. Multiple claims on a given resource will result in multiple claims in the catalogue; the resource will be eligible for deletion when all claims are expired. The space management service uses claims-catalogue data and DDM tools to clean up (delete) data for which claims are expired (or, if necessary, to apply a prioritization algorithm to delete live claims to sustain a high watermark level of disk space). The space management service can also check consistency and integrity of the catalogue by comparing catalogue and SE contents (a spider-like function). The space management service can operate either as a persistent daemon or as a periodically submitted batch job.

Sites will need a local catalogue of locally resident data blocks and their local state (completeness, disk-resident, etc.). The claims catalogue can fulfil this need.

4.6.5.2 Data Movement

All managed data movement in the system is automated using the subscription system described. Replication tools can also be used directly for automated, reliable transfer of individual datasets or files, e.g. by end users, but subject to usage and resource controls. Data movement operations should aspire to *most* data movement being done in a managed way by production, physics working group or regional operators, with end-user data movement a minor contributor. The system must nonetheless offer convenient and robust end-user data movement, including support for 'off-Grid' endpoints such as laptops.

Managed data movement proceeds through block transfers triggered by subscription fulfilment. Data movement is triggered from the destination side, such that local uploading can be done using site-specific mechanisms if desired, with no requirement that other sites be aware of these specialized mechanisms. A number of independent entities per site are involved in the block movement process to perform the principal steps of the movement process: fetching of the set of logical files to be transferred on the basis of the block content minus any files already present locally; replica resolution to find the available file replicas via the dataset location catalogue, content catalogue and local replica catalogues; allocation of the transfers across available sources according to policy; bulk reliable file transfer; and file- and block-level verification of the transfer.

Multi-hop transfers will sometimes be required. These will be supported via chained subscriptions through the intervening hop sites. For example if for efficiency or bandwidth reasons transfers between CERN and site X should proceed via Tier-1 centre Y, then Y will hold a subscription to the CERN resident data and X will subscribe specifically to Y's instance such that X receives files via Y.

The block transfer service utilizes two site-local catalogues for managing transfers. A *local content catalogue* recording the files still needing to be transferred to complete the block, and a *queued transfers catalogue* acting as a state machine for active file transfers (see the implementation discussion below).

4.6.5.3 Scalable Distributed Usage and Partitioning

With the exceptions of the local replica catalogue and space management service, the catalogue components described are global in scope. We foresee replicating copies of global system catalogues (or subsets of them) down the tier hierarchy as read-only replicas in order to offload central services for usages able to tolerate slightly 'stale' data. We are working with the 3D project to put in place the infrastructure to manage the replication, updating and use of remote replicas. Writes will be done only to the central master instances.

A further means of managing load and increasing scalability is partitioning of global system components through the use of multiple instances with distinct scopes. This will certainly be done for the dataset content catalogue, as discussed above. It may be done for other components as well, motivated not just by scalability but also by managing access rights, as discussed in the next section.

4.6.5.4 Local Component Instances and Cloning of the DDM System

We foresee also that distinct local instances of at least some components will be required, e.g. a local instance of dataset catalogue infrastructure to define and manage private or temporary datasets that define data aggregations used by local production operations and/or local analysis users, and a local subscription infrastructure to support data distribution for production and regional dissemination of data (e.g. Tier-1 to Tier-2s). The catalogue infrastructure foresees 'publishing' metadata to distinguish strictly local content from content that can be marked ready for publication upwards to the global ATLAS system.

As a more extreme case, we also foresee the possibility of complete clones of the full DDM system. A Tier-1 centre, for example, may want to install and operate an instance of the full DDM system, not coupled to the ATLAS-wide system, with the scope of the clone system being the service region of the Tier-1. This instance could be used for data management of resources 'private' to the region independently of the management of ATLAS-wide resources.

4.6.5.5 Roles, Authentication, Authorization

Use of the system normally requires Grid authentication in order to establish virtual-organisation identity and role(s). All data in the system is readable by all ATLAS VO members, but not by others. Physical files on storage systems are owned by the DDM system (a DDM userid), with Unix read permission granted to a Unix group encompassing ATLAS users. All writes and deletes to managed data areas are managed by DDM system processes.

Roles needed are reader, writer and administrator. These are the roles which will be supported for database accesses within the authentication infrastructure being developed by the 3D project, so assignment of these roles to relational catalogues in the system will be supported. A user identity, determined typically from a Grid certificate, is mapped to the appropriate DB roles based on VOMS (or equivalent) information and a policy module managing role mapping for a particular database.

By default, all authenticated ATLAS users hold the reader role for all data. The writer and administrator roles are assigned to users for specific contexts/subsets of the system for which they are entitled to such access. Where the system catalogues are partitioned along functional lines, this contextual role assignment can happen at the catalogue level. For example, a physics-working-group (PWG) dataset repository (distinct catalogue instance) could have administrator

rights assigned to the PWG leader and writer rights assigned to PWG members, while the repository for main production would grant these roles to the production system operators.

It will not be practical in all cases to partition the system such that user IDs map to roles at the whole-catalogue level. An example is user-level catalogues: the system must support user-level capability to define and work with datasets, but a catalogue infrastructure per user would be impractical. In this case, an authenticated user must be allocated write/administrator rights only to a subset of the catalogues 'owned' by the user. This can be implemented in a file catalogue supporting a Unix-like hierarchy and ACLs on 'directories' which can correspond to a user directory. The LHC File catalogue (LFC) supports this functionality, and will be tried out as a means of supporting this case

4.6.6 Implementation

The implementation approaches being pursued for the components of the first DQ2 version are described here. Implementations may evolve in subsequent versions depending upon the success of the initial implementation and the availability of new third-party software.

A common element in many of the component implementations is the use of the POOL File Catalogue (POOL FC), both its interface and its implementation. The functionality of many of the catalogues in the system is a close match to that provided by a file catalogue (basically, one-to-many string lookup plus associated metadata). In view of this, we use the POOL FC interface for these components in order to leverage a robust existing component for which many relevant back-end implementations are available. Furthermore, we use POOL's own FC implementation in many of the present prototype components, since it offers ease of use, robustness, high performance, and a useful range of back end technologies (Oracle and MySQL via RAL, XML).

4.6.6.1 Dataset Repository

The repository is implemented using the POOL FC. Given a dataset ID or name, it returns information on the dataset and provides navigational information to dataset constituents (logical files, other datasets). The FC FileID corresponds to dataset ID (DSID), and LFN corresponds to dataset name. Additional metadata attributes include version, mutability state, creation date, expiration status, publication status (local, global, private, ready for publication), and provenance reference. A supplementary table records dataset versions (DSID, version ID, version-specific metadata)

4.6.6.2 Dataset Hierarchy

The dataset hierarchy augments the dataset repository with information on the hierarchical structure of container datasets, and supports versioning of container datasets. A possible implementation approach, still to be tested, is to use the Hierarchical Versioning System (HVS) developed by ATLAS and LCG AA.

4.6.6.3 Dataset Content Catalogue

The dataset content catalogue is implemented using the POOL FC. Several instances are used, partitioned on the basis of dataset metadata to achieve scalability. The FC FileID corresponds to

dataset ID (DSID), LFN corresponds to constituent LFN, and PFN corresponds to constituent GUID. Additional metadata attributes include dataset version number and file-content metadata (e.g. event count)

4.6.6.4 Dataset Location Catalogue

The dataset location catalogue is implemented using the POOL FC. For a given dataset, it returns a list of site identifiers holding a copy. For immutable datasets (data blocks) it acts as a catalogue of block locations. The location catalogue is also used to implement subscriptions: a site registered as a holder of a DUID (dataset-unique ID) for a mutable dataset holds a subscription to the dataset, and new versions (identified by version-unique IDs, VUIDs) are delivered to the site. A site holding only a specific version of a mutable dataset is registered as a holder of the VUID rather than DUID. The FC FileID attribute corresponds to dataset ID (DUID or VUID), LFN corresponds to dataset name, and PFN corresponds to site identifier. Additional metadata attributes include version/completion status, date, owner, and availability (resident at site, or accessible from site).

4.6.6.5 Claims Catalogue

The claims catalogue is implemented using the POOL FC. Separate instances record dataset claims and file claims. The FC FileID attribute corresponds to claim ID, LFN corresponds to owner, and PFN corresponds to dataset or file resource that is claimed. One claim can reserve multiple resources. Additional metadata attributes include claim category or role level (user, group, system), creation time, originating processor/job, lifetime, and status (pending, granted, terminated, expired).

4.6.6.6 Local Replica Catalogue

The local replica catalogue is implemented using the POOL FC. The back-end implementation that is used may be Grid or site dependent, but must support the POOL FC interface and meet performance requirements. The catalogue provides logical-to-physical file lookup for files at the local site (or in the region of the catalogue's scope). It provides both GUID and LFN mappings to physical files.

4.6.6.7 Mass Store Interface

ATLAS has requested via the LCG that sites offering mass storage make their services available via uniformly robust and functional SRM interfaces offering the functionality of SRM 1.1 plus (if possible) the space reservation features of SRM 2. Other functionality of SRM 2 and SRM 3 is not required.

4.6.6.8 Data Movement

Present tools for data movement are GridFTP and SRM, used directly by an in-house reliable file transfer (RFT) system in DQ. DQ2 will take the same approach if robust RFT systems are not available from middleware providers, but we are first seeking a middleware RFT system that will meet our needs. The LCG/EGEE (gLite) FTS is currently under evaluation.

Data movement is driven from the destination side. Transfers typically have a gsiftp: or srm: source and a destination that is either one of these or something site-specific. All sites must provide external access via gsiftp and/or srm. Within the site, site-specific transfer or upload mechanisms can be used.

A transfer task first scans the local file catalogue to determine whether any requested files are already present (e.g. due to file overlap with a block already present). If any are found, claims are asserted on those files to protect them from deletion. Also, claims are asserted on source data block(s) prior to commencing the transfer, to protect from deletion during the transfer.

A state machine is used to persistently record transfer status for all files being moved by all active replication tasks at the site. If two tasks are replicating the same files (fulfilling subscriptions on datasets with overlapping content), the state machine will ensure that overlapping files are transferred only once.

Data sources are discovered by using the data location features of the architecture: the data location catalogue resolves the available site locations for a dataset, the dataset content catalogue provides LFN content, and the replica catalogues of the site or sites selected (on the basis of cost/efficiency) as data sources are used to resolve LFNs into physical files (storage URLs). A replication task will not necessarily use only a single source: if a data block to be replicated has instances at several accessible sites, the replication task can specify a prioritized list of sites to try as sources, and the prioritization can differ file to file in order to spread the replication task across (functional) sites.

4.6.7 Organization of System Development, Deployment and Operation

DQ2 system development takes place within the DDM subproject of the ATLAS database and data management project. This activity encompasses

- Design, development, testing, maintenance of the DDM system
- Liaison and collaboration with contributing external projects and regional Grids
- Support services for production and end-user usages of the system

System development responsibilities are shared between the core ATLAS data-management team and regional Grid participants (who may also be core team members). The core data-management team has primary (but not exclusive) responsibility for

- Overall design of system architecture, components, interfaces
- Definition of performance metrics and benchmarks to be used in initial implementation decisions
- Specification and development of the general and generic parts of the system not specific to a particular Grid
- Specification and as-needed development of the 'baseline implementation' of the full system for use in contexts (such as the Tier-0) where ATLAS fully controls the implementation choices. This baseline may or may not coincide with one of the 'Grid flavor' implementations, and it may or may not be adopted by or co-developed with one or more of the Grid flavors.

Regional Grid participants have (over and above any core team responsibilities if they are core team members) responsibility for

- Input on requirements and design of overall system to ensure compatibility with the regional Grids
- Benchmarking potential regional implementation choices against the established metrics for required performance
- Specification and as-needed development of the Grid-specific components and implementations required to integrate the regional Grid into the overall ATLAS system. (Contrast with DQ, in which the core developer has had the responsibility for developing the Grid-specific implementations.)

Deployment and support for the relational database services upon which many DDM components rely is provided by the database services activity discussed below, with the assistance and support of the DDM system development team. Operation of the DDM system is the responsibility of the ATLAS Computing Operations organization. DDM system developers provide direct support to operations and end users on usage of the system, debugging, feedback gathering and improvements, etc.

4.6.8 DQ2 Development and Deployment Timeline

The development and deployment timeline for DQ2 is as follows:

- 2005 Q2 – system design, initial component evaluation and selection, prototype implementation
- 2005 Q3 – performance and scalability tests of a functionally complete prototype, deployment for commissioning, development iteration based on testing/deployment experience with the prototype
- 2005 Q4 – full production deployment on Service Challenge 3 service. In production use by production system and end users.

4.7 Book-keeping for Production and Analysis

4.7.1 Production Database

The production database is the persistent back-end to the ATLAS production system, which is described in detail in Chapter 5. The present production database was developed as part of the production system development effort for DC2. The database currently holds information about tasks (aggregations of jobs), datasets (aggregations of logical files), task transformations (specifications describing the transformation of datasets into other datasets), job transformations (an executable transforming logical files into other logical files), job definitions and job executions (information about every attempt at executing each job).

After almost one year of operation the production database holds 74 job-transformation records, ~1000 tasks and ~2500 datasets, 1 million job definitions, 1.6 million job executions, and about 2.3 million logical files. The size of the database is less than 4 GB.

The database is implemented with Oracle on the physics DB service of CERN IT. In February 2005, after chronic performance problems, it was moved from the general physics cluster to a

dedicated server. Performance with the dedicated server has been adequate, but the server load caused by the production activity has been extremely high (I/O rate from disk greater than 100 MB/s, equivalent to reading in the complete DB every 40 seconds). Work has been under way with the assistance of IT experts to understand the reasons for this load and optimize the database and its client applications. This has led recently to a reduction in the load by more than an order of magnitude. The dedicated server should now be able to sustain a load ten times higher than the current production activity.

The production system architecture supports the physical distribution and/or replication of the database, while presenting to its clients the appearance of only a single production database. This keeps the system simple and scalable. It is expected to be straightforward to use this capability to introduce logically several production databases.

By its nature the production database will grow monotonically. However the part of it that is in active use (jobs yet to be processed, in processing, or recently processed) will remain relatively constant, only growing with the CPU capacity available to the experiment. We expect the database back-end to be able to exploit this 'sliding hot data' pattern, with old records archived so as not to impact hot-data access. Access costs for archived data will be higher, but accesses will be infrequent.

The production database contains a wealth of information useful for computing derived statistics such as number of jobs executed per day, daily failure rates, specific-failure incidence evolution over time, correlations, and so on. Extraction of such monitoring statistics must not interfere with the ongoing production activity. At present, apart from exceptional incidents, indications are that these two activities can coexist. To ensure this remains the case, we expect that in the near future the derived data will be computed only once and stored in additional tables. These tables could then easily be moved to another server if necessary to exclude the possibility of interference altogether.

Further development of the production database will take place in the context of production system development, in close collaboration with the database and data management project. An important development activity will be integration with the DDM system, in particular with the DDM system's management of dataset and logical file information. At the start of production-database development it was not clear what file metadata would be useful, and whether the Grid-native metadata catalogues would allow schema evolution in a reasonable way. Hence it was decided to simply store the file metadata in the production database, under our full control. The file metadata problem is now being addressed in the context of the DDM system, and we expect that the additional copy in the production database will become unnecessary.

4.7.2 Offline Book-keeping

ATLAS has used the AMI database framework for offline book-keeping in DC1, DC2, and for the 2004 combined test beam. AMI is designed to be very flexible; it supports different schema, and different relational database backends. This feature has been extremely useful as requirements evolve, and as different and complementary tools have become available. AMI-compliant databases are self-describing, which means that the same software can be used as usages and schema evolve. In particular a generic and configurable web search interface gives access to all data in AMI-compliant databases.

Although MySQL was exclusively used by AMI up to April 2005, an Oracle back-end has been successfully integrated in a development version during May 2005. It is expected to go into production during the summer of 2005.

AMI has been available as a web service since December 2003. Clients have been developed in several languages; the most frequently used being Java and Python, but C++ is also available. The Python client is supported by the GANGA (Python-based interactive interface to Grid services) team, and work is under way to use it for the input of book-keeping data from Athena job options files.

AMI implements a fine-grained authorization system which allows mapping of users to different roles. Integration of VOMS authentication will soon be available.

ATLAS has chosen to keep the production system database, which describes the production jobs, completely separate from the physics metadata book-keeping. The latter system, for DC2 in particular, has suffered from a lack of integration in the production system, impairing the transmission of physics-relevant metadata from the production system to the physics book-keeping database. Considerable work has been done to improve the links between these two systems and an interface for requesting production tasks has been developed. Thus in the future it should be much easier for non-expert users to submit jobs to the production system, with propagation of complete and coherent dataset descriptions to the book-keeping database. We expect this interface to further improve in the next generation of the production system now beginning development.

In the context of the ATLAS Distributed Data Management architecture now under development, AMI will provide the dataset selection catalogue (Section 4.6.5.1.5). The major project for AMI will be to add a specific production and analysis web-search interface to complement the current generic one. This interface will be designed following consultation of the physicist user community. It is expected that this interface will not only provide searches on the data contained in the AMI databases themselves, but will also serve as a portal towards other database services such as DQ2 or COOL.

4.7.3 Provenance and Related Metadata

Provenance information is maintained at many scales. Each event "knows" its parent (the RAW event used to produce the ESD, the ESD event used to produce the AOD, and so on), and retains a pointer to it whenever the parent is stored in object format: this is the means by which back navigation is supported. Similarly, the book-keeping system tracks which input file was used to produce a specific output file, along with data describing the transformation used to accomplish this.

The control framework group will deliver in 2005 a "history object" framework encompassing job, service, algorithm and data object histories that will allow one to record and later discover details about the algorithm that created a given physics data object and the job in which that algorithm ran. Persistence of this history information in the ATLAS event store will provide a means to record and query data provenance at the sub-event ("How were these tracks produced?") level.

Other metadata related to provenance include information such as the geometry version used in a simulation, or the calibration tags used in reconstruction. The book-keeping system tracks such things at the dataset level, so one can configure a client job appropriately, though work re-

mains to support dynamic runtime extraction of such information from book-keeping and metadata databases. At the individual event level, an extensible TagInfo object currently records the current geometry tag and version tags from the Athena IOVDbSvc, the service that handles access to time-varying data such as conditions and calibrations.

4.8 Database and Data Management Services

4.8.1 Distributed Database Services

The distributed database services activity in the DB project provides the physical database services and supporting software (e.g. client connection software) making up the distributed database infrastructure supporting database-based ATLAS applications, including conditions, geometry, DDM and other databases.

ATLAS is addressing this area through a combination of in-house effort and close collaboration with the LCG. ATLAS was instrumental in the establishment of the Distributed Deployment of Databases (3D) project in LCG [4-10] to address the facilities and software needs of establishing the database service, replication and scalable access infrastructure required to support DB applications. To meet these needs ATLAS relies heavily on and collaborates with 3D. ATLAS database applications are based on centralized writing and distributed reading, with no multi-master requirements on the DB services. DB inputs originating far from the centralized services are written to the central services, either via remote write or by first transporting the inputs close to the centralized services. This approach avoids the complexities of multi-mastering by leveraging the fact that for all our DB applications, writes are much less frequent and have lesser scalability requirements than reads.

The common LCG database replication framework under development by the 3D project will be used for data distribution to Tier-1s and Tier-2s. Tier-1s will typically offer Oracle and possibly also MySQL services. For Tier-2 support in particular, distribution/replication mechanisms must be as automated as possible because of the low support levels available. ATLAS will deploy replica servers to Tier-2s using centrally issued Grid job-submission commands that fully automate the process. Thus, no local site administration will be necessary.

Both Tier-1 and Tier-2 servers will be continuously replicating data published on the master servers at CERN. We expect on the order of 100 servers to be deployed and maintained across all major Tier-2 sites. In addition, some support is likely to be required for smaller local sites that wish to deploy replicas. For all replica servers, automatic updates from a central source will be possible. For replication management and monitoring, ATLAS will use a suite of tools now under development by the 3D project as well as tools we have for ATLAS-specific use cases.

Tier centres will in general not have to manage their conditions-data selection policy locally. Following the ATLAS Computing Model, Tiers do not locally control the ATLAS-wide production processed at them. ATLAS-level processing management, through automated mechanisms, will drive the population of needed conditions data sets at the local resource based on the processing allocated there. Likewise they will not have to locally manage the database storage issues. Several mechanisms are available or being explored to distribute conditions data subsets of interest: selective replication out of Oracle masters into MySQL or SQLite replicas; tiered DB access with caching intermediaries (the FroNtier approach); and data block subscriptions for file-based data. We expect to employ all of these in appropriate contexts.

However, we further expect that some Tier centres will want to have some control over what conditions data they get. There will be calibration activities centred around particular Tier-2s, e.g. an institute or group of institutes which has responsibility for calibrating a particular sub-detector will want its conditions data to do analysis and preparation of new calibrations. To support this we will have conditions data 'streams', analogous to event-store data streams. These streams represent subsets of the whole conditions database, and particular Tiers will 'subscribe' to particular conditions streams. This implies distribution of both the IOV/relational database data associated with a conditions stream, and any corresponding conditions data payload files (e.g. POOL ROOT files). It is similar to sites that 'subscribe' to particular classes of event datasets, a concept being built into the distributed data management model, and the same mechanism will support this case. A further use case for 'subscription' replication to tiers is the distribution of larger conditions data payloads that are stored in files.

4.8.2 Caching Tiers for Distributed Database Access

From our experiences to date, relying on database replication alone to achieve scalability would require substantial manpower and logistics both centrally and at Tier centres. We are in the planning stages of pursuing an alternative approach that would reduce or eliminate the need for wide dissemination of database replicas, while providing fully automated local caching of needed conditions (and potentially other) data. The approach is that of the FroNtier project developed at FNAL for Run II experiments [4-9], in which client/server communication is via conventional HTTP, with the URL constituting the upstream message and containing the specification of the requested data, and the page content of the returned document (prepared by a web application server interacting with the database) packaging the requested data in a format understood by the client application. Because the protocol is that of the web itself, standard web proxy caching tools (such as Squid) can be used to establish automatic caching of requested data close to the client, e.g. on gateway servers at a Tier site or even a contributing site not affiliated with ATLAS.

A number of issues need to be addressed, in particular the form and granularity of the URL-requested data packets and a mechanism to prevent retrieval of stale data from the cache. Studies are planned soon by 3D and COOL to examine the compatibility of FroNtier itself with the COOL interface. Approaches to the cache validity problem are being explored. For example, any incorporation of data packet version information into the request will ensure the retrieval of the correct packet. A two-stage retrieval could support this. In the first stage a request goes to the IOV database for the data packet valid at a specified time (this exchange is not cached, since the request is too specific to have a significant probability of reuse, but it would also return only a small message). The data message returned would contain URL(s) for the specific, versioned data packets required to fulfil the request (this is an example of the so-called REST approach to web-based services, the architectural model used by the web itself). The URL or URLs would be requested in the second, cached, step (and any other local process requesting calibration data in a similar time frame will have a high probability of a cache hit on these data packets). CDF tests show orders of magnitude performance improvement due to local Squid data caching with this approach.

The 3D project provides the context for the common effort on applying this approach for the LHC. We are also examining other open-source projects for potentially relevant work in managing cache validation in tiered data exchanges on the web.

4.8.3 Operational Experience

Operational experience in database services during DC2 has identified a number of performance bottlenecks including server capacity limitations and concurrent DB-connection count limits. Another related performance limiter was TCP/IP socket timeouts and failures on the client side. This condition (similar to a denial-of-service attack) happens in the case of high query rates and particularly affects geographically remote sites. For all these cases, deployment of more geographically distributed database server replicas resolved the problem.

As expected, we found that the deployment of replica servers is an effective tool towards achieving scalable database services. To ease the burden of replica-server administration and support on remote sites, the database-server replica installation was reduced to a one line command. However, we have found that only the centres that have problems with access to the central databases (because of firewalls or geographical remoteness resulting in low data throughput) deployed the replica servers. In addition, numerous concerns were expressed regarding replica synchronization with the central servers. To address these, the new deployment model involving centralized distribution of replicas via Grid commands is being prototyped, which will enable centralized, coherent management of the distribution and synchronization of replicas. A proof-of-the-principle Grid-based replica server deployment has been demonstrated at the SMU site of Grid3.

An alternative to Grid deployment of replicas on site head nodes is delivery of database services directly to worker nodes, using conventional Grid transport mechanisms for services installation. During ATLAS DC1 this approach was extensively tested on NorduGrid and remains a deployment option. It is particularly attractive for sites where worker nodes have no outbound connectivity.

4.8.4 Monitoring

During ATLAS Data Challenges and other large-scale production efforts we have deployed (and steadily improved) several database monitoring tools. The most detailed information has been collected through server-side logging. This information has been crucial in debugging database service problems and errors reported by clients. To complement this server-side data, an ATLAS Client Library is being developed to provide monitoring and logging information on the client side.

In addition, higher-level monitoring information of server status has been collected both on the server side and remotely by database monitoring servers that collect and present global information about all ATLAS database servers. The operating system level information was collected with the help of the LEMON monitoring services provided by the CERN/IT FIO group and the Ganglia monitoring services at BNL. Database server level information was collected via the dedicated open-source monitoring tools deployed outside of CERN.

Experience shows that experiment-specific monitoring tools are important for efficient monitoring of database services. We are pleased with progress in monitoring capabilities for ATLAS-specific needs that has been provided by the central physics database services at CERN. As these monitoring needs are shared between all the LHC experiments, we anticipate that the 3D project will be a vehicle to provide effective, distributed database-services monitoring capabilities for ATLAS. The 3D project is evaluating the client-side monitoring system developed at Fermilab for monitoring Oracle-query performance.

In planning for database services capacities for robust Grid operations one has to provide for the maximum load. Traditionally, in a controlled centralized environment of organized data processing these capacities are easy to calculate and provide for. The much less predictable environment of distributed processing on a federation of Grids makes provision of required capacity a challenging task. It is important to study the collective dynamics of the "chaotic" workload exhibited in this environment in order to understand load behaviours and optimize both the provisioning of the services and their utilization. To this end we have deployed load-monitoring services for our databases and are using them to track collective usage, adjust server deployments and usage patterns, and study workload patterns.

The collective dynamics of database access were studied in DC2. Each computational task that retrieved input parameters required by the job accessed the same database. With thousands of tasks accessing the database daily, the daily average database server load was 122 queries per second with a standard deviation of 0.24 of the daily average value, which is fourteen times more than purely statistical fluctuations. We believe these large fluctuations reflect a high degree of correlations among the computational tasks running on the three Grids used by ATLAS, which together encompass more than ten thousand processors (e.g. jobs starting simultaneously on many nodes of a processing farm). Monitoring and analyzing such fluctuations will give us guidance in developing database deployment and production-usage approaches that can minimize them, while at the same time allowing us to track service needs and growth.

4.8.5 Roles

In accordance with established best practices adopted in ATLAS, users will be granted only those database-access rights required for their work, as determined by their role. To this end we envisage separation of user roles in database access, with access privileges granted according to the following roles: administrators, developers, editors, writers and readers. Administrators manage the installation of database servers and give access rights to other users. To develop database applications for particular software domains, e.g. online, calibrations database, geometry database, and so on, developers are granted full access rights to a particular database. The editor role covers cases in which UPDATE or DELETE privileges are required. For improved operational security the main applications or scripts that write data to the database are granted only the INSERT and SELECT privileges, so that the data – a growing collection of records – cannot be deleted when a writer account becomes compromized. The most frequent database operation is performed in the reader role that has only the SELECT privilege.

It is expected that only a small number of users will have administrative rights on production servers. The number may be larger on development servers. We are investigating long-term solutions for password management; in particular, a database authentication mechanism based on Grid proxy certificates that eliminates the need for clear-text passwords.

4.8.6 Database Authentication

Several approaches exist to provide database services in the Open Grid Services Architecture (OGSA). A key element in the architecture is the Grid security model providing database-access control. Two implementation options have been identified. In the first option a separate server (e.g. LDAP) performs Grid authorization on behalf of the database server. The alternative is to do Grid authentication and authorization in the database server itself, which avoids significant data transfer overheads inherent in the multilayer authentication option. The second option has

been implemented in an ATLAS Grid-enabled MySQL server prototype, avoiding the use of clear-text passwords and eliminating significant encryption and cryptographic handshake overheads generated when a stateless protocol is used. The prototype was successfully tested with certificates issued by DOE, CERN and NorduGrid certificate authorities, and was tested in ATLAS Data Challenge 1 production for authentication and database write-access control. This experience has been promising, and further work is under way to expand backward compatibility of Grid proxy tools and to add the server purpose information to Grid host certificates. Work is under way in 3D to implement the first option using an LDAP server together with Oracle. Oracle itself is exploring the second option, but the timescale is unclear.

An extended Grid security model will benefit from the introduction of shared Grid certificates (similar to the privileged accounts traditionally shared in HEP computing for production, librarian, data management and database administration tasks) as well as tools for establishing a Grid proxy upon login (similar to the AFS token acquisition procedure).

4.9 Development Processes and Infrastructure

Database and data management development in ATLAS follows the procedures and processes of the Software Project's Software Infrastructure Team. Software is maintained in the ATLAS offline software repository. Software coupled to the ATLAS offline framework, such as Athena persistency software, is incorporated and distributed with ATLAS offline releases. Software without this coupling, such as the distributed-data-management system DQ2, or the software configuration system (CMT), is not included in offline releases, but is distributed using the same Pacman-based distribution mechanisms as other ATLAS software, and will appear as ATLAS software 'project' components when the project partitioning of ATLAS software is completed.

ATLAS databases share a pattern of discrete, regular releases with the experiment software, but it is a distinct release schedule not necessarily coupled to software releases. Accordingly, the DB project is putting in place a 'database release coordinator' role and activity to manage the complex coordination of database releases that are consistent both internally and with prevalent software releases.

Testing and quality assurance must have a high priority in this domain. Databases and data management are heavily exposed to the sheer scale of ATLAS computing, so scalability and performance testing are vitally important. Data storage is also key to long term readability of data, so regression testing is essential. In coming scalability tests during the computing commissioning and service-challenge programs, as well as in routine production, ATLAS DDM will be deploying systems designed and intended for good performance at full startup scales, and will be evaluated and improved accordingly based on these smaller scale deployments – while smaller scale, we will evaluate whether system scalability shows comfortable headroom to reach data-taking scales, and adjust design and development accordingly. Testing of data storage and database software systems are integrated with the nightly and larger scale (RTT) automated testing programs of ATLAS software. Dedicated regression tests monitor readability of old formats and check for inadvertent schema changes.

4.9.1 Milestones, Planning, Manpower

Milestones and deliverables for the DB project are maintained here [4-14]. Project manpower is tracked at the subproject level. Currently about 42 people are contributing materially to the project, with a very approximate 25 FTEs of total effort. An essential factor in being able to deliver ATLAS needs with available manpower is leveraging external and collaborative work, in particular the LCG AA efforts in POOL, COOL, ROOT and SEAL; the CERN IT Physics Database team; the 3D project in LCG GD; and the Grid projects. While Grid projects are well staffed, the LCG AA, IT Physics DB, and 3D teams are not; we rely on robust support for all these projects to complete the ATLAS system. Within ATLAS, manpower is adequate in most areas but there are two very important exceptions. Database services is a large and rapidly growing activity, essential to ATLAS data processing, and is presently severely understaffed. Oracle database administrator (DBA) expertise is almost completely lacking in ATLAS and must be addressed with the hire of at least one ATLAS Oracle professional DBA. We expect these issues to be addressed, respectively, by the establishment of an ATLAS Computing Operations organization and by the availability of computing infrastructure support funds through M&O following the MoU process.

4.10 References

4-1 F. Glege, *The Rack Wizard - a graphical database interface for electronics configuration*, paper presented at LECC 2003 - 9th Workshop for Electronics for LHC experiments, Amsterdam, 29th Sept - 3rd Oct 2003.

4-2 ATLAS collaboration, *High Level Trigger, Data Acquisition and Controls TDR*, CERN/LHCC/2003-022, page 167.

4-3 I. Soloviev, *OKS users guide*, EDMS note ATL-DQ-ES-0051 (2002).

4-4 RD45 project: *A Persistent Object Manager for HEP,* http://wwwasd.web.cern.ch/wwwasd/cernlib/rd45/

4-5 A. Amorim et al., *An Implementation for the ATLAS Conditions Data Management based on Relational DBMSs*, IEEE Trans. Nucl. Sci. 51 (2004) 591.

4-6 See for example http://atlas.web.cern.ch/Atlas/GROUPS/LIQARGON/Data_Bases/LArNOVA_v0.2.html

4-7 A. Amorim et al., *Conditions Databases, the interfaces between the different ATLAS systems*, paper presented at CHEP04, Interlaken, Swizerland, September 2004.

4-8 ATLAS offline book-keeping database AMI: http://isnpx1158.in2p3.fr:8180/AMI/

4-9 Frontier project: http://lynx.fnal.gov/ntier-wiki

4-10 LCG distributed deployment of databases (3D) project: http://lcg3d.cern.ch/

4-11 A. Valassi and V. Tsulaia, *Hierarchical versioning component for the conditions database*, http://wwwlistbox.cern.ch/earchive/project-lcg-peb-conditionsdb/doc00000.doc

4-12 Enhydra Octopus replication software, Object Web Consortium, http://www.objectweb.org

4-13 M. Branco, *ATLAS DonQuijote data management system*, https://uimon.cern.ch/twiki/bin/view/Atlas/DonQuijote

4-14 ATLAS database project schedules, http://atlas.web.cern.ch/Atlas/GROUPS/DATABASE/project/mgmt/

5 Grid Tools and Services

5.1 Introduction

The ATLAS Computing Model foresees the distribution of raw and processed data to Tier-1 and Tier-2 centres, so as to be able to exploit fully the computing resources that are made available to the Collaboration. Additional computing resources will be available for data processing and analysis at Tier-3 centres and other computing facilities to which ATLAS may have access.

A complex set of tools and distributed services, enabling the automatic distribution and processing of the large amounts of data, has been developed and deployed by ATLAS in cooperation with the LHC Computing Grid (LCG) Project and with the middleware providers of the three large Grid infrastructures we use: EGEE, OSG and NorduGrid. The tools are designed in a flexible way, in order to have the possibility to extend them to use other types of Grid middleware in the future. These tools, and the service infrastructure on which they depend, were initially developed in the context of centrally managed, distributed Monte Carlo production exercises. They will be re-used wherever possible to create systems and tools for individual user access to data and compute resources, providing a distributed analysis environment for general usage by the ATLAS Collaboration.

This chapter describes the relations between the ATLAS Computing system, the LCG and the Grid middleware projects, and the requirements placed on those projects. It also describes the architecture, current implementation, and planned future developments of the ATLAS distributed production and analysis systems. The first version of the production system was deployed in Summer 2004 and used in the second half of 2004. It has been used for Data Challenge 2 (DC2), for the production of simulated data for the ATLAS Physics Workshop (Rome, June 2005), and for the reconstruction and analysis of the 2004 Combined Test Beam (CTB) data.

5.2 Relations with the LCG Project and with Grid Middleware Providers

Computing resources can be accessed by ATLAS physicists through Grid middleware components and services. The principles and objectives of computational data Grid technology aim to provide capabilities that are well suited to the distributed operation foreseen by the ATLAS Computing Model. These include services for software installation and publication, production operations and data access for analysis through a uniform security and authorization infrastructure, interfaces for remote job submission and data retrieval, and job scheduling tools designed to optimize utilization of computing resources.

The LHC Computing Grid (LCG) Project [5-1] was set up at the beginning of 2002 to deploy and operate Grid middleware for the LHC experiments, and to help them to integrate Grid tools with their software base. Currently the LCG project has a dual role:

1. it acts as an umbrella organization for all Grid middleware deployments that are relevant for LHC experiments;
2. it deploys and operates one particular middleware suite (LCG-2 and later updates) in cooperation with the European Union funded EGEE middleware project.

Three "flavours" of Grid middleware are deployed on computing resources that are available to ATLAS: Grid3/OSG in the USA, NorduGrid/ARC in Scandinavia and a few other countries, and LCG-2/EGEE in most of Europe, in Canada and in the Far East. In order to make optimal use of all resources, the ATLAS production and analysis systems are designed to be independent of any particular Grid flavour; only the few software components that access directly specific Grid tools are Grid-flavour dependent.

However, as ATLAS computing must work across the Grid deployments in a coherent and sustainable way, it is vital that:

- the Grid deployments used by HEP ensure the highest possible degree of interoperability at the service and API level;
- that changes in those services and APIs be advertized well in advance, and technical support be made available to experiment developers where applicable.

It is expected that a major focus of the LHC Computing Grid project will be in promoting the interoperability between the various Grids and encouraging convergence wherever possible. It will also have a role in developing a common layer above the various Grid deployments if that is needed. At present, such layered tools are being developed within the experiments, but common tools developed in a timely fashion would enhance sustainability.

ATLAS has strong representation in the LCG organization, and participates actively in the monthly meetings of the LCG Grid Deployment Board and the weekly operations meetings organized by the Grid Deployment Area. For example, the ATLAS Computing Coordinator sits on the LCG Project Execution Board. ATLAS collaboration members sit also on the management boards of the three Grid middleware development projects, and guarantee a constant flow of information in both directions.

5.3 Integration of Grid Tools

5.3.1 The ATLAS Virtual Organization

It is assumed that all members of the ATLAS Collaboration are authorized to become members of the ATLAS Virtual Organization (VO), and therefore are allowed to submit jobs and gain access to Grid resources. While currently it is not a trivial operation to submit jobs to the Grid, and it has effectively been done only by production managers, we have to foresee facilitating access to the Grid for analysis users.

In order to open access to Grid computing facilities to all members of the ATLAS Collaboration, we are setting up an internal system of priority allocation for the usage of CPU and data storage resources. The problem is multi-dimensional, as one can define at least three dimensions that can affect the priority of a job on a given site:

1. The role of each person in the computing structure:
    - Grid software administrator (who installs software and manages the resources);
    - production manager, who will have higher priority than normal users for "official" group productions and will be able to place files in commonly accessible areas;
    - any normal user.

2. The activity (physics, combined performance, detector groups and central computing operations);
3. The funding agency or institute affiliation of the person submitting the job.

The VOMS (Virtual Organization Management Service [5-2]) middleware package allows the definition of user groups and roles within the ATLAS Virtual Organization. We are initially setting up a VOMS group for each Physics Working Group, Combined Performance group and Detector System, as well as a generic ATLAS group and other groups for software testing, validation and central production activities.

At the time of registration with the ATLAS VO, a user will indicate the group (or groups) with which he or she is going to work, and in which capacity. The VO manager will check with the group leaders that the assignments and the proposed roles are correct.

In order to implement an efficient allocation sharing and priority system in the Grid access tools, it is necessary that the use of the resources dedicated to ATLAS be prioritized according to the Collaboration's policy. This could be achieved in principle via a central job distribution system to fully-dedicated clusters, but such a solution may suffer from bottlenecks and single points of failure. In a distributed submission system, local resource providers will have to be involved in defining and enforcing policies on priorities (in agreement with the Collaboration) through pseudo-dynamic updates to the local resource management systems fair-share policies. These options are under investigation as part of this year's upgrade plan (see Section 5.4.7).

In this context, one important issue to be addressed in the near future is the mapping of Grid users to local group accounts, through which tools it happens and how local priorities are taken into account. Identity mapping is necessary when a site's resources do not use Grid credentials natively, but instead use a different mechanism to identify users, such as UNIX accounts or Kerberos principals. In these cases, the Grid credential for each incoming job must be associated with an appropriate site credential.

The Open Science Grid (OSG) in the US has set up the GUMS (Grid User Management System [5-3]) project to develop a Grid Identity Mapping Service. The GUMS server performs this mapping and communicates it to the gatekeepers. The gatekeepers are charged with enforcing the site mapping established by GUMS. The GUMS client (gatekeeper) portion consists of an administration tool for querying and/or changing the state of the server. While this tool is not yet of general use outside OSG, it could in the future be adopted more widely to manage local heterogeneous environments with multiple gatekeepers; it allows the implementation of a single site-wide usage policy, thereby providing better control and security for access to the site's Grid resources. Using GUMS, individual resource administrators are able to assign different mapping policies to different groups of users and define groups of hosts on which the mappings will be used. GUMS was designed to integrate with the site's local information services (such as HR databases or LDAP).

5.3.2 Data Management on the Grid

The data management system fulfils a set of distinct functions: global cataloguing of files, global movement of files, and space management on ATLAS-owned storage resources. In ATLAS we opted initially to realize the global catalogue function by building on the existing catalogues of the three Grid deployments (the Globus RLS in the case of NorduGrid and Grid3, the EDG-RLS in the case of the LCG). The data management system acted as a thin layer channelling cata-

logue requests to the respective Grid catalogues and collecting/aggregating the answers. At the same time it presented the users with a uniform interface on top of the native Grid data-management tools, both for the catalogue functions and the data movement functions. The current implementation of the data management system used in ATLAS [5-4] and its foreseen evolution are described in Section 4.6.

Following the experience of Data Challenge 2 in 2004, we have re-examined the basic assumptions of this system. A large fraction of job failures were due to lack of robustness of the native Grid tools used for data cataloguing and data transfer, as well as to a lack of appropriate Storage Element space-management tools. We had to implement many work-arounds to offset the middleware problems (such as retries and time-outs in file transfers, resubmission of jobs to different sites when input data became unavailable etc.), and at the same time we asked middleware providers to develop more robust data-management systems.

In order to build a robust Distributed Data Management system for ATLAS, we request that:

- all sites deploy Storage Elements presenting the same interface (SRM), backed up by a reliable disk pool or mass storage management system, such as DPM, dCache or Castor;
- Grid middleware implements an infrastructure for storage quotas, based on user groups and roles;
- Grid middleware implements hooks for file placement policies depending on users' roles;
- all mass-storage-based SE's provide information on file stage-in status (disk or tape) to be used for job scheduling and file prestaging.

The data management system based on native Grid catalogues has proven not to scale to the current needs of the ATLAS experiment. We are therefore in the process of redesigning the system, basing it on a multi-tiered cataloguing system. The basic concept is to have each dataset (a coherent set of events, recorded or selected according to the same procedure) subdivided into a number of “data blocks”. Each data block can consist of many files, but is considered a single indivisible unit in terms of data storage and transfer. Central catalogues need to know only the location of the Storage Elements where each data block is present (in addition of course to the relevant metadata); the mapping to the physical filenames is known only to local catalogues. All local catalogues are required to present the same interface to the ATLAS data management system, but could in principle be implemented using different Grid-specific technologies. More information on the new ATLAS Distributed Data Management System (DDM) can be found in Section 4.6.6.

5.3.3 Job Submission to the Grid

ATLAS has gained experience of job submission through Data Challenges 1 and 2, and through its preparations for the Rome Physics Workshop in June 2005. At present the job-submission system works with LCG2, Grid3, NorduGrid, and is starting to gain experience with gLite. It has also worked with the precursors to these Grid deployments. The system is now able to describe the function of a generic job submission, and to draw out some additional requirements.

5.3.3.1 Generic Job Submission to the Grid

An ATLAS Collaboration member is assumed to have already joined the ATLAS VO and to have been granted a certificate to be used for job submission. Tools on a user-interface machine

then allow the user to generate, from this certificate, a delegated proxy. This is sent along with the job, enabling the job to perform Grid operations on behalf of the submitter.

In addition, in order to submit a job, the user must provide a job-description file (created either by hand, or via automatic tools like a Production System executor, or directly from the user interface tools), in which all the information about the job is specified. It includes the executable to be run and its parameters, the files to be staged to the execution host, the requirements and the preferences for the execution host (e.g. available memory, maximum CPU time granted to a job etc.) and the files to be staged out. Requirements and preferences concerning the system on which the job will run and the execution of the job itself are also specified (typically by expressions evaluating to a boolean or a numerical value).

When ready, the submission system sends the job-description file – together with all the specified accessory files – to a Grid service that verifies the user credentials, takes care of finding a suitable resource and dispatches the job to it. The submission command returns a job identifier, which is unique to the job; this is used to reference the job in all the subsequent commands.

Using other tools and the job identifier, the user can poll the status of his or her job, delete it if desired, or retrieve the output of the finished job by transferring it to the submitting machine. In case of errors, the user can inspect the job history by querying a logging and book-keeping service, where the various middleware components push a log of the operations they perform on the job.

5.3.3.2 Requirements and Desirable Features for Job Submission

The job-management techniques described above are rather low-level, and are seldom to be used directly by the user, other than for occasional submission or for debugging purposes. More often, the users implement their own tools, from simple scripts to large applications like the production system, to manage large quantities of jobs in a transparent way. For this reason, the user interface must provide also an API. Additionally, multiple-job submission with common required files and trivial variations in parameters should be possible through a single submission by the user.

In order to be able to optimize job submission and distribution on the Grid, a certain number of middleware services must be provided. First of all, one needs a reliable information system, so that jobs can be submitted to sites that support ATLAS, have the required system and software configuration, and have available capacity and network connectivity. Other workload-management services may be present, such as the Resource Broker on the LCG Grid. In all cases these services must be extremely reliable and scalable with a large number of jobs and/or bulk job submission.

Information for job monitoring, accounting, and error reporting must be provided consistently by Grid middleware. Jobs must never disappear without a trace. It is highly desirable that the reporting use a uniform schema for all Grids. In particular, Grid middleware must:

- have the local job identifier returned at time of submission to the submitter, to allow easy access to job status and match of log files;
- make access to the job's standard output possible while the job is running, to give single users as well as production managers a way to find out what happens in specific cases;
- report errors as they occur and not hide them behind generic messages (such as "Maximum Retry Count exceeded").

It is highly desirable that the user can specify job groups and be able to tag jobs at submission time, in order to interact with and monitor those specific jobs amongst the many they may have submitted.

Job distribution on the Grid will be done by a combination of Grid services (such as the Resource Broker for the LCG Grid) and directives given by the submission system. In order to optimize job distribution and minimize data transfer and network activity, it is desirable to have ways to send jobs “reasonably close” to where their input data reside. An exact, and dynamically updated, implementation of a connectivity matrix between all Computing Elements and all Storage Elements is not needed, as network bandwidths can differ by orders of magnitude between local clusters and trans-oceanic links. Instead, a simple matrix of the "distance" between CEs and SEs, even in rough terms (local, close, same continent, faraway), could be used in match-making and improve, at least on average, network traffic and job throughput.

5.4 The Distributed Production System

5.4.1 Architecture

The production system distinguishes between two levels of abstraction (see Figure 5-1). On the higher level, input datasets are transformed into output datasets by applying a task transformation. The process of doing this is called a task.

Datasets are usually quite large and consist of many logical files. At a lower level of abstraction, input logical files are transformed into output logical files by applying a job transformation, as shown in Figure 5-1. This process is called a job.

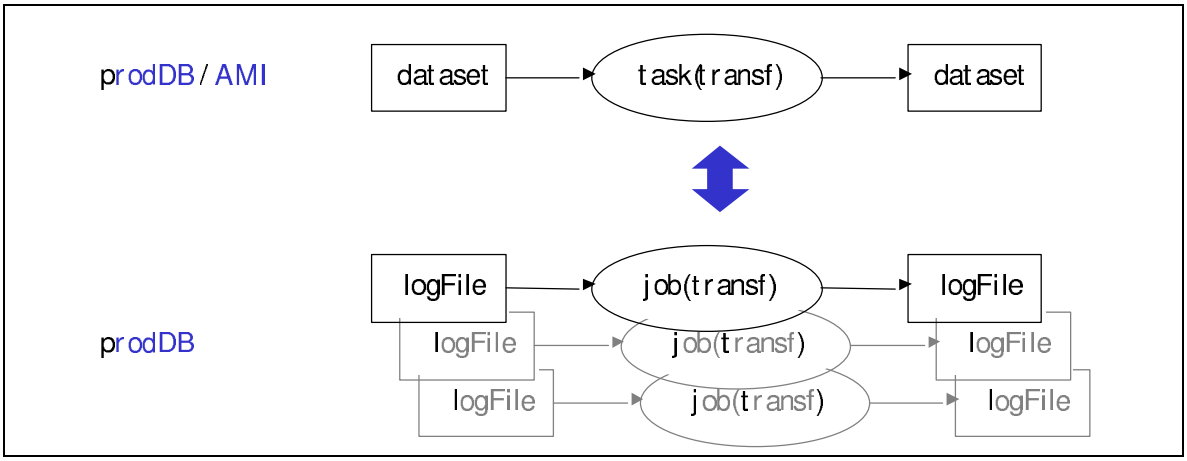

Figure 5-1 Conceptual view of datasets and transformations.

The architecture of the production system was designed in mid-2003 with the following goals in mind:

- The system should be as flexible as possible while retaining simplicity.
- The system must target automatic (i.e. with minimal human intervention) production on each of the three Grids in use within ATLAS in 2004, i.e. LCG, NorduGrid and Grid3. Legacy batch systems must be supported as well as a back-up solution.

- The system should use Grid middleware services as much as possible.

The resulting design is shown in Figure 5-2. It is based on four components: a central production database (ProdDB), a data management system (dms), a supervisor component, and an executor component. In the following subsections we will address each of these components separately.

By adopting a component-oriented design and additionally allowing the components to run as agents on different servers, communicating asynchronously with each other, a lot of flexibility, both with respect to the logical composition of the system and to its physical realization, was made possible.

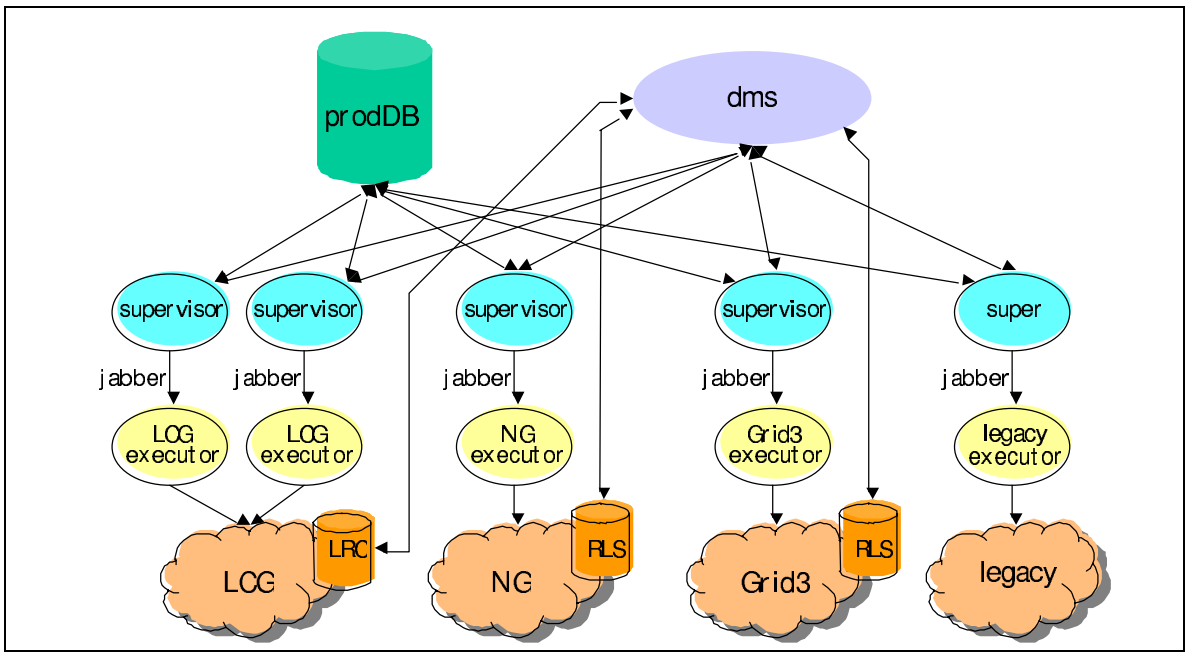

Figure 5-2 Production system architecture.

Note that the design assumed that each Grid system knows how to manage its workload and collect information about its own system. This had the advantage that we did not have to invest effort in producing our own equivalents of this functionality. However, it also made the overall production system reliant on the performance of the individual Grid systems, which was found not to be adequate in some cases. Of course Grid systems are not yet perfect, but the production system has only limited possibilities to compensate for such imperfections.

5.4.2 Job Transformations

The job transformation is the script that sets up the run-time environment, allows possible compilation of patches to software in a release, runs the Athena executable, parses the log file for known warning and error messages, and tidies everything up at the end. Currently it is implemented as a shell script. It can include any data file that may be missing from the release or that may be needed for a particular job.

There are two kinds of transformations: production (system independent) and KitValidation (KVT, test oriented). They differ in the call method, with positional arguments for production

and switched parameters (name-value pairs) for KVT. Building the transformation for production jobs is at present a manual operation; the transformation is then put into a Pacman cache and loaded by Grid jobs at start-up.

Work is in progress to provide a generic transformation for all usages, including non-production jobs. The intention is to move to Python (instead of shell) scripts, as in this way the transformation can be seen as an extension of the jobOptions. The transformation will also check the integrity of the expected output file at the end of the job. (This needs the knowledge of the number of output events as counted inside the Athena job.)

5.4.3 Production Database

There is only a single logical production database. The physical realization of the database may be distributed and/or replicated, but to the other components in the design it will present itself as a single entity.

The database holds tables with records for:

- job transformations;
- job definitions;
- job executions;
- logical files.

A job-transformation record describes a particular combination of executable and release. The description includes the signature of the transformation, listing each formal parameter together with its type (restricting the possible values) and its meta-type (indicating how the values should be passed to the executable).

Each job-definition record points to its associated job transformation. Other fields allow one to keep track of the current attempt at executing this job (lastAttempt), which supervisor component is handling this job (supervisor), what is the relative priority of this job (priority), etc. The bulk of the job definition is, however, stored as an XML tree in the field jobXML. It lists the actual values to be assigned to the formal parameters of the transformation and additional information about logical input files and logical output files.

For each job definition there can be zero, one, or more job-execution records, corresponding to each attempt at executing the job. Each attempt has a unique number which is appended to the names (both logical and physical) of all files produced, ensuring interference-free operation even in the case of lost and/or zombie jobs. The execution record also records information like start- and end-time of the job, resources consumed, where the outputs were stored, etc.

In the last table, logicalFile, the production system stores all metadata about logical files. Most of the information is redundant with respect to the information stored in the respective metadata catalogues of the Grids (size, guid, md5sum, logicalCollection), but at the time the production system was developed these metadata catalogues did not support schema evolution and ATLAS did not know *a priori* what metadata was needed. Consequently, it was decided to deploy temporarily our own catalogue in addition to filling and using the existing ones.

The production database used in 2004–2005 for Data Challenge 2 and subsequent productions was implemented as an Oracle database hosted at CERN. A MySQL version of the production database is also available for small-scale productions.

5.4.4 Supervisor

The next component, called the supervisor, takes free jobs from the production database and hands them on to one of the executors with which it is connected. The information about jobs is exchanged using XML, usually wrapped in XMPP (using the Jabber protocol) or wrapped in SOAP using web services. As its name suggests, the supervisor will follow up on the job, asking at regular intervals about the job status until the job is declared 'done' by the executor. At that point, the supervisor will, for successful jobs, verify the existence of all expected outputs, and, if all is as expected, will rename them to their final logical name (by dropping the attempt number from their temporary logical name). Additionally, the files will be added to the logicalFile table together with any metadata produced by the job and returned by the executor. In the case of a failed job the supervisor will simply release the job in the production database, so that it can be picked up again if the maximum number of attempts is not yet reached.

The supervisor does not perform any brokering. The handing-out of jobs is based on a simple "how-many-do-you-want" protocol. The supervisor asks the executor how many jobs it wants (possibly qualified with resource requirements) and the executor replies with a number (possibly qualified with, not necessarily the same, characteristics). The supervisor may then send a number of jobs to the executor, which in turn may choose to process or refuse them. The non-binding nature of the protocol allows both very sophisticated and very simple implementations to co-exist on both the executor and supervisor side.

For efficiency reasons an implementation of the supervisor can keep state but the design does not require this. Having a stateless component obviously makes it more resilient against crashes.

The supervisor implementation used in 2004–2005 is Windmill [5-5]. Each Windmill instance connects with a specific executor, and manages all jobs processed by this executor. Since Windmill is stateless, it can be robustly reconnected to the same executor without loss of job-status information.

5.4.5 Executors

The task of the executors is to interface the supervisor to the different Grid or legacy systems. They translate the system-neutral job definition into the system-specific language (xrsl, jdl, wrapper scripts, etc.), possibly adding some pre- and post-processing steps like staging in/out of files. The executors implement a system-neutral interface with the usual methods: submit, getStatus, kill, etc. Again, the design does not require the executor to keep state.

Five executors were developed and deployed in 2004–2005 and used for Data Challenge 2 and subsequent productions:

- Dulcinea [5-6] for the NorduGrid;
- Capone [5-7] for Grid3;
- Lexor [5-8] for the LCG Grid;

- Lexor-CG for the LCG Grid as a test of direct job submission to the Computing Elements;
- Bequest [5-9] for batch systems like LSF, PBS, BQS.

5.4.5.1 Dulcinea

The Advanced Resource Connector (ARC) is a Grid middleware suite developed by the NorduGrid collaboration. NorduGrid's ARC has been deployed at a number of computing resources around the world. These resources are running various Linux distributions and use several different local resource-management systems (LRMS). Despite using different LRMS and specific information providers, the different sites can present the information about their available resources in a uniform way in ARC's information system. This information system is based on the Globus Monitoring and Discovery Service (MDS). The information provided is used by the brokering algorithm in the ARC user interface to find suitable resources for the tasks to be performed, as well as by various monitoring tools like the NorduGrid Grid Monitor. NorduGrid's ARC is fully integrated with the Globus Replica Location Service (RLS) [5-10]. This allows jobs sent to an ARC-enabled site to specify input-file locations as an RLS catalogue entry instead of a physical-file location. A job can also, if desired, automatically register created output files in an RLS catalogue.

The ATLAS production system executor for NorduGrid's ARC, Dulcinea, is implemented as a C++ shared library. This shared library is imported into the production system's Python framework. The executor calls the ARC user-interface API and the Globus RLS API to perform its tasks. The job description received from the Windmill supervisor in the form of an XML message is translated by the Dulcinea executor into an extended resource specification language (XRSL) job description. This job description is then sent to one of the ARC-enabled sites, selecting a suitable site using the resource-brokering capabilities of the ARC user-interface API. In the brokering, among other things, the availability of free CPUs and the amount of data needed to be staged-in on each site to perform a specific task is taken into account.

The look-up of input-data files in the RLS catalogue and the stage-in of these files to the site is done automatically by the ARC Grid Manager. The same is true for stage-out of output data to a storage element and the registration of these files in the RLS catalogue. The Dulcinea executor only has to add the additional RLS attributes needed for the DonQuijote data management system to the existing file registrations.

The Dulcinea executor also takes advantage of the capabilities of the ARC middleware in other respects. The executor does not have to keep any local information about the jobs it is handling, but can rely on the job information provided by the Grid information system.

Because of the diversity of the ARC-enabled resources in terms of which Linux distribution is installed on the various sites, the ATLAS software release is recompiled for various distributions. The ATLAS software release is then installed on the sites that want to participate in ATLAS productions. After testing the installation at each site, the ATLAS run-time environment is published in the site's Grid information system. By letting the Dulcinea executor request the run-time environment in its job description, only sites with the proper software installation are considered in the brokering.

The ARC middleware and the Dulcinea executor provided a stable service for ATLAS DC2 in 2004. Twenty-two sites in seven countries operated as a single resource and contributed approximately 30% of the total ATLAS DC2 production. Most of the resources were not 100% dedicat-

ed to the DC2 production. The number of middleware-related problems was negligible, except for the initial instability of the RLS server. Most job failures were due to site-specific problems.

5.4.5.2 Capone

The Grid3 executor system, Capone, communicates with the supervisor and handles all the interactions with Grid3 resources and services. Job requests from the supervisor are taken by Capone, which interfaces to a number of middleware clients on the Virtual Data Toolkit (VDT). These are used with information in Grid servers to submit jobs to the Grid.

ATLAS job parameters are delivered to Capone in an XML-formatted message from the supervisor. These messages include job-specific information such as specific parameters expected by the transformation (the executable), identification of input and output files, and resource-specific requirements for the job. Capone receives these messages as either Jabber or Web service requests and translates them into its own internal format before beginning to process the job. The transformation to be executed is determined from the input data contained in the supervisor message. The Globus RLS (Replica Location Sever [5-10]) is consulted to verify the existence of any required input files and serves metadata information associated with these files.

The next step is to generate a directed-acyclic graph (DAG) which is used to define the workflow of the job. The job can now be scheduled using a concrete DAG generated from the previous abstract DAG with the addition of defined compute and storage elements. The Computing Element and Storage Element are chosen among those in a predefined pool of Grid3 resources. To select the resource, it is possible to use different static algorithms like round-robin, weighted round-robin or random, and some dynamic ones that account for site loads.

The submission and subsequent monitoring of the jobs on the Grid is performed using Condor-G [5-11]. On the CE, if necessary, all input files are first staged-in before the execution proceeds. The ATLAS software itself, an Athena executable, is invoked from a sandbox using a VDT-supplied executable. A wrapper script, specific to the transformation to be executed, is called first to ensure that the environment is set up correctly before starting any ATLAS-specific executables. The wrapper script also checks for errors during execution and reports results back to the submitter. In addition, the wrapper script performs additional functions such as evaluating an MD5 checksum on all output files. The Condor status of each job is checked periodically by Capone and, when the remote job completes, Capone resumes the control of the job.

Capone checks the results of the remote execution: the program's exit codes and the presence of all the expected output files. This is a delicate step since a large variety of errors may be discovered here, ranging from IO problems encountered during the stage-in process, to errors in the execution of Athena, to site characteristics that prevent Condor from exiting correctly. The next step towards job completion is the stageout of the output files that are transferred from the data area in the CE to the output SE. The transfer also involves evaluation of the MD5 checksum and file size of the destination file(s), to check the integrity of the transfers. Furthermore, some important metadata, like the GUID (a globally unique file identifier), is recovered from service files in the remote-execution directory. Finally there is the registration of the output files to RLS inserting logical filenames, physical filenames and metadata attributes required by DonQuijote [5-4], the ATLAS data-movement utility that allows data transfers between Grids.

Over the course of three months, July–September 2004, about one third of ATLAS DC2 production was executed on Grid3, keeping pace with peer projects using NorduGrid and the LCG. Efficiencies steadily improved as lessons learned at production scales were incorporated into the

system. We found that two areas of Grid infrastructure are critical to distributed frameworks: job-state management, control and persistency, and troubleshooting failures in end-to-end integrated applications. The next steps in the project will make progress in those areas. The evolution of Capone is towards increasing its reliability, for example, implementing robustness to Grid failures, and making it more flexible to support user-based production for distributed-analysis jobs.

5.4.5.3 Lexor

Lexor is the executor that submits jobs to the LCG-2 Grid. Lexor has been fully coded in Python, as this language encourages a modular, object-oriented approach to application development. First, a module implementing a generic LCG job class was developed. The code is based on the SWIG API to the workload-management system, developed by the Work Package 1 (WP1) of the European DataGrid (EDG) project for their User Interface. Through this API, this class offers methods for defining, submitting, and managing the job on the Grid. For job definition, a set of methods allows the manipulation of the underlying JDL description, either by direct access to the classAd attributes or via higher-level functions. Other methods are used to tune and perform the job submission, to monitor the job status (and extract single pieces of information from the status), and finally to retrieve the output sandbox of the job. A job-cancellation method is also provided, but is not yet used in the production system.

When it is in submission mode, the supervisor periodically asks for the number of jobs the executor is willing to manage. To answer this question, we issue a query to the LCG Information System to retrieve the number of free CPUs in the sites matching the requirements expressed in the supervisor request. Computing this number turned out to be a tricky task, because of the aggregated nature of the information published by the LCG Information System. Often we overestimated the real amount of available CPUs, because some of the advertized ones are not accessible by ATLAS jobs. This problem was temporarily addressed by defining one queue for each VO, but a refining of the information schema has to be considered.

Once submitted, a job is dispatched to the Computing Element (CE) selected by the broker and queued until the local batch system starts executing it on a Worker Node (WN). Before the required transformation can be started, several actions have to be performed on the WN, so the job has to be wrapped by an appropriate script. Typical actions are the interactions with external services, such as the repository where the transformation packages are hosted, or the Don-Quijote server. The wrapper also embedded mechanisms to cope with temporary problems in the underlying Grid. A frequent problem is the occasional unavailability of ATLAS software, due to NFS problems between the CE exporting the software directory and the WN. Jobs being executed on such hosts fail at the very beginning, thus freeing a WN that attracts more and more jobs. To avoid this “black-hole” effect, the wrapper was modified to have the job send an e-mail message to the submitter about the problem, and then sleep for some hours. When it wakes up, the process is repeated until the software is found or the job is killed. This keeps the WN busy, preventing further attempts to run jobs on it.

The stage-in and the stage-out phases of the wrapper were also enclosed in a *sleep-and-retry* cycle. The LCG commands used to stage files are in fact very sensitive to failures or long delays in the services they rely on. The heavy work-load, imposed on these services by the high number of concurrent requests from the running jobs, sometimes caused them to become unresponsive for a while or even to crash. This affected all the running jobs, which, after having successfully executed the transformation, failed to stage the produced files to the output Storage Element. The retry mechanism doesn’t completely solve the problem, but it allows the recovery of many

jobs where the LCG commands hung. In fact, by killing by hand the command process, a subsequent attempt is triggered, which usually succeeds.

During the DC2 production in 2004, Lexor managed more than 100 000 jobs, including generation, simulation, digitization and pile-up. More than 30 LCG sites were involved, providing overall about 3 000 CPUs. The job management was shared, on average, among six Lexor instances owned by different users, each instance taking care of a maximum of about 800 jobs at a time. This threshold was decided in order to limit the size of the queries issued by supervisors to the central production database and to circumvent limitations of the resource broker.

5.4.5.4 Lexor-CG

Lexor-CondorG (Lexor-CG) was developed from the original Lexor executor to address the problem of slow job-submission rates to LCG resources. It was observed that the interaction with the LCG WMS at that time took up to 45 seconds per job, and this was too slow to keep the available CPUs filled. A simple calculation, assuming 4000 available CPUs and jobs lasting 8 hours, illustrates that a submission rate of at least 1 job per 7 seconds is required. Indeed, during the Rome production (early 2005) there were eight instances of Lexor required in order to approach this submission rate. The resource requirement, in terms of hardware, services (RBs), and particularly manpower, was deemed not to scale, and a different submission solution was sought.

CondorG is standard Grid middleware for remote job submission to CEs, and indeed it forms part of the LCG RB. In this case the RB chooses the destination CE, and CondorG submits to the named site. However, when given information about the resources, CondorG can also do the resource brokerage. This information is taken from the BDII and converted into the Condor ClassAd format.

The fundamental difference, compared with the original Lexor executor, is that the resource brokerage and the submission are done by separate components. The Negotiator and one or more Schedulers run on different machines, and scalability is achieved by increasing the number of Schedulers only. Furthermore, the scheduler is sufficiently lightweight to run much closer to the UI, perhaps on the same machine.

The interaction with the local Scheduler is therefore much faster, and job submission takes 0.5 s (cf. 45 s). Similarly the status and getOutput requests are instantaneous as the response is like that of a local batch system.

There are, however, two perceived deficiencies with this approach. First, if the UI machine hosts the Scheduler then it cannot be turned off, which is inconvenient if the UI machine is, for example, a laptop. For the Rome production this issue did not arise because the Windmill services must keep running on the UI. In general the UI could be a high-availability machine to which a user can connect with "ssh" to submit jobs. In terms of fault-tolerance, the CondorG scheduler will regain its state after a reboot. Lastly, CondorG does provide for separating the UI and Scheduler functionality – it is the Condor-C implementation that is included in the recent EGEE/gLite release. This option has not yet been tested.

A second concern was the lack of central logging and book-keeping (L&B) when using CondorG. We should stress "central" because there is in fact a local record of the stages in the job's life, and a mechanism exists to extract this to a MySQL database. The LCG central L&B has been identified as a potential cause of the poor performance, so not having this architecture is an ad-

vantage of CondorG. This does not prevent the L&B information being migrated to a central place, asynchronous to job submission.

Lexor was used as the basis for the Lexor-CG executor because its modular design allowed the easy exchange of the LCG submission with the CondorG submission. Everything else, including the run scripts, stage-in, stage-out, validation, etc. remained the same and was re-used. During production operation, improvements to Lexor were also applied to Lexor-CG.

The method of choosing how many jobs to submit to LCG was different. The number of free CPUs published on the Grid was thought often to be inaccurate and therefore best ignored. Instead jobs were submitted as quickly as possible until the queues were loaded to a configurable depth and then submission was throttled. This was achieved by placing the following in the job requirements: "*...&& WaitingJobs+CurMatches < 0.1*TotalCpus*". So for a 100 CPU sites, only 10 jobs would be queued, after which the site would fail the requirements. Jobs that cannot find a matching site will remain in the queue UNMATCHED. When the number of UNMATCHED jobs reaches a configurable number, then the executor submission is throttled.

This exhibits two features of CondorG not available in the LCG RB. CurMatches is a per-CE attribute that is incremented by 1 for each match made to that CE. When the information is refreshed it is reset to zero. This enables the matchmaking to handle stale information sensibly and leads to a round-robin job distribution amongst sites with similar ranks. The LCG RB tends to submit blindly to the highest-ranking site, until the information updates. This led to a slug of jobs going to each site and very poor job distribution in terms of queuing jobs unnecessarily.

The input and output sandbox mechanism had to be provided by the CondorG executor. The GRAM protocol ensures that the executable reaches the worker node, and that the standard output and error gets back to the submission machine. So we built the executable to be a self-extracting tarball containing the input sandbox. Similarly the output logs are tarred into the stdout. This provides the same functionality as the LCG RB sandboxing, but minimizes the number of file transfers, which for the RB each require two hops with time-consuming authentication and authorization.

The Lexor-CG executor began submitting jobs part way into the Rome production of early 2005. The production rate on LCG was immediately doubled, because of the full use of LCG resources. This was maintained over the course of the Rome production and two Lexor-CG instances, for fault tolerance, were able to match the production rate of eight plain Lexor instances. This was directly attributable to the higher submission rate.

During the reconstruction phase it was noted that having more executors was an advantage. The shorter jobs, with four output files requiring renaming and validation, meant that Windmill was now the bottleneck. Removing the job-submission-rate bottleneck was an important step towards identifying this, and it will be addressed in the future production system.

5.4.5.5 Bequest

The legacy executor Bequest is designed primarily to be simple and provide an adaptable system which may be ported to any batch flavour. In spite of this aim for simplicity, it is inevitable that different environments and different batch systems deployed at sites will require some reconfiguration of the executor. The development aims to keep this reconfiguring to a minimum. The Legacy executor aims to fit seamlessly into the production system providing the standard

interface to the supervisor and to be indistinguishable, to all intents and purposes, from a Grid-implemented executor.

Bequest was developed using Python, thus allowing for rapid development and testing. The supervisor project development was also undertaken using Python and provides a simple executor template which was utilized during the development. In addition, several supervisor tools for the interpretation, processing, and creation of XML messages were also used by Bequest. The executor allows some site-specific configuration values to be set, such as the batch queues to interface to, and the maximum number of executor jobs allowed into these queues.

In addition to ensuring that Bequest provided the functionality to interface with the supervisor and run jobs on a local batch system, it was considered a design goal to ensure that it also provide an adequate level of persistency and fault tolerance. The executor is made persistent by storing the global-to-local job mapping and allowing queries access to this store. Thus when the executor fails, a persistent store of job information remains intact, and upon re-launching the executor the correct information about the assigned jobs can be obtained. This persistency is very important in a distributed system to ensure that failures in the system don't cause the loss of jobs at remote sites.

Bequest was primarily developed with a PBS batch system at the FZK centre in Karlsruhe, Germany. At various points during the development process the executor was deployed at alternate sites and adapted to their systems. Sites used include RAL (PBS) and CERN (LSF). Once a working framework was available, a parallel development was undertaken using the CCIN2P3 (BQS) system. The deployment at different sites allowed the executor to remain focused on its goal of providing a generic system with minimum required alterations for different sites.

The PBS-based executor for the FZK site was tested extensively with both event-generation and simulation jobs. In the final testing phase the executor was run with a continuous float of 200 jobs, with 400 jobs, with a typical duration of around 24 hrs each, being processed during this test. Although this number is somewhat smaller than the numbers available when using Grid-based executors, it is nevertheless convincing for a single batch system, and it is envisaged that the executor would be capable of scaling to even larger batch-system capacities.

The persistency and resistance to executor/supervisor failure was tested by killing either the executor or supervisor once jobs had been submitted to the batch system. The majority of jobs could be successfully retrieved, although in some cases the jobs were lost. More investigation is required into the mechanism by which jobs were lost, and to ensure that this possible cause of failure is corrected.

5.4.6 Experience with the Production System

Between the start of DC2 in July 2004 and the end of September 2004, the automatic production system submitted about 235k jobs belonging to 158k job definitions, producing about 250k logical files. These jobs were approximately evenly distributed over the three Grid flavours. The definitions belonged to 157 different tasks, exercising 22 different transformations. Overall, they consumed ~15 million SI2k months of CPU (~5000 CPU months on average present-day CPUs) and produced more than 30 TB of physics data.

By design, the production system was highly dependent on the services of the Grids it interfaced to. This was known to be a risky dependency from the beginning and indeed we suffered considerably because of it. The Globus RLS deployed by both NorduGrid and Grid3 turned out

to be very unstable and became reasonably reliable only after a series of bug fixes. We had a similar experience with several of the LCG services, e.g. the resource broker and the information system. Because the LCG is by design the most complex system of the three Grids, requiring many services to work at the same time, it is not surprising that the LCG had the highest failure rate of the three Grids.

Transient site misconfigurations and network failures were amongst the most frequent causes of job failures. The correctness of published information, some of which had to be handled manually at least at the beginning of DC2, turned out to be a very important factor in job and data distribution and in the efficient use of the available computing resources.

It was not only the Grid software that needed many bug fixes. The data challenge started before the development, let alone the testing, of the ATLAS production system was finished. As a result, various bugs had to be corrected, and new features introduced, during the data challenge period.

5.4.7 Evolution of the Production System

The experience with the distributed production system accumulated during Data Challenge 2 in 2004 was analysed through a review in January 2005 [5-12]. The bulk of the recommendations of the review panel were addressed by setting up a "Grid Tools Task Force", active in February and March 2005, which in turn produced guidelines for the "2nd version" of the production system, an action list, and an implementation plan [5-13]. While both the review panel and the task force concluded that the high-level architecture of the production system is basically sound, many of the underlying implementations have to be made more robust in order to shield ATLAS from the possible shortcomings of Grid middleware and of running conditions.

5.4.7.1 Production Database

Tests during February 2005 of a dedicated Oracle database server for the ProdDB were positive, therefore the production was moved to the new dedicated Oracle server in March. After the move, the database response times were measured in production conditions, for queries submitted by supervisors and monitoring activities. The immediate effect was in an improvement by 2–3 orders of magnitude for some of the common queries, but there are some queries (logically not very different from other ones) that take an almost infinite time to get executed. It also appears that the whole database is read every few minutes. As the database is now dedicated, we could activate logging of all queries from the database side with the aim of finding out which queries are performed routinely on the database, by whom, and how heavy they are; investigations are in progress with the help of IT Oracle experts.

If after all these investigations the response time is still found to be inadequate, then, depending on the reason found from the query logs, we can improve indexing of the database, create numeric fields for commonly accessed string (or XML) fields, or investigate modifying the schema to have separate tables for finished and active jobs, so as to reduce the number of accessed records and separate production from accounting activities.

In addition, we have to make sure that there are adequate fields in the database tables to store the quantities needed for monitoring and accounting.

The use of the production database (or an extension thereof) for "user productions" and analysis jobs has been discussed during last the few months. Separate instances of the production database, implemented in Oracle or in MySQL, have been used to submit moderate-sized productions to the Grid by physics groups, with acceptable performance. Currently these additional activities are possible only because they are limited in number and in scope, as there is no way to assign relative priorities to jobs that are submitted to the Grid through separate systems.

On the other hand, having a single production database for all ATLAS Grid jobs may not scale and may introduce a single point of failure in the system. The solution (currently under study) may be in a system where resource providers take an active part in the definition of job priorities, allowing jobs to be submitted by different systems but with a different level of access to local resources (see Section 5.3.1).

5.4.7.2 Supervisor

The Windmill supervisor, developed in 2003–2004 for DC2 productions, had to cope with many more error conditions than anticipated. Its functionality had to be extended with respect to the original design, and at the same time its complexity increased considerably. The work model also changed during the early phases of DC2: we went from having a single instance of the supervisor that communicates with several instances of the executor, to running supervisor–executor pairs on the same machine. This change made the remote communication protocol between the supervisor and the executor, based on the Jabber protocol and Jabber servers, unnecessary.

Eowyn [5-14] is a new implementation of the supervisor component. The most important difference with respect to Windmill is the introduction of an additional interface layer between the supervisor and the executor that exchanges information using Python objects and defines a synchronous communication protocol that allows executors to answer immediately if they can. Underneath this layer the communication can still be implemented using asynchronous XML message exchanges (e.g. using Jabber), but in addition executors can also be plugged directly into the supervisor. Existing executors can work with Eowyn using an adaptor implementing the expected time-out functionality, and translating between Python objects and XML. For executors implemented in Python and translating the XML back into objects, it would of course make sense to eliminate this overhead.

Eowyn also reduces the number of queries to the production database as it keeps the state of active jobs in memory, in addition to keeping it in the production database. In case of a restart of an Eowyn-executor pair, the in-memory database is restored from the production database with a minimal amount of queries.

The Grid Tools Task Force [5-13] recommended continuing testing Eowyn and completing its implementation, with the aim of having it in production in summer 2005. The proposed modifications of interfaces from Windmill to Eowyn are under discussion in order to agree them with the authors of all the executors. Tests of Eowyn are starting from systems that are less dependent on Grid middleware: first Bequest, then Lexor-CG, and finally all other executors.

5.4.7.3 Executors

Existing executors have also been discussed by the Grid Tools Task Force [5-13]. An internal code review took place in this context, where authors of one executor looked in detail at the im-

plementation of another. This exchange of experience resulted in concrete suggestions to improve the quality and above all the robustness of the code. The level of commonality between the Python-based executors will also increase.

Capone, the Grid3/OSG executor, has to provide a substantially larger functionality than the other executors, as Grid3 does not provide a job brokering and distribution system. The Capone team are working with the VDT developers in the US to move part of the current Capone functionality to future Grid3/OSG middleware. The aim is to simplify Capone for ATLAS users by transferring as much as possible of the complexity to Grid middleware.

The "agent" model, pioneered by LHCb with their Dirac production system [5-15], has been very effective for their organized productions in 2004–2005. Within ATLAS we have started exploring the strengths of this model and the possibilities of implementing it. The model consists in distributing a large number of identical jobs ("agents") that just set up the running environment, contact a central database ("task queue") and get a job (or jobs) to be executed locally. In this model all brokering is effectively done on the experiment's side. A central task queue would make it easier to assign priorities to jobs but could constitute a serious single point of failure; more complex schemes with several task queues (perhaps one per group or activity as defined in VOMS) are possible in principle but have not been investigated yet.

5.4.7.4 Monitoring and Accounting

Like any system that performs complex tasks, the Grid has to expose its status to the end user, to allow for real-time monitoring and long-term performance analysis based on accumulated data. In what follows, we shall distinguish between two observation methods:

- Monitoring: real-time access to dynamic information about the current status of the system and the progress of tasks it performs, for purposes of troubleshooting, debugging etc.
- Book-keeping: persistent storage of historical information for purposes of statistical analysis, accounting, auditing etc.

Both kinds of information relate to the same objects and their attributes, such that information collected by monitoring tools can be naturally re-directed for further book-keeping. However, the overlap is not complete, as some information is volatile and has no value in a historical perspective (e.g., proxy expiration time). Moreover, it is physically impossible to store all possible data persistently; hence, for book-keeping purposes, only a selected subset of monitored parameters is chosen.

Accounting is then based on the information provided by the book-keeping system.

5.4.7.4.1 Monitoring

By "monitoring" we refer both to "job monitoring", i.e. checking the progress in the execution of single jobs, and "Grid monitoring", i.e. checking the status of the Grid, including the availability of resources and our job distribution.

Specifics of monitoring are:

- freshness: the presented information must be an up-to-date snapshot, with the returned data being no older than a few minutes, or even seconds – depending on the nature of data;

- promptness: a system must be capable of answering monitoring queries within seconds – or well within the information-refresh period;
- scalability: a system must scale both with number of servers and number of clients.

Monitoring tools do not require a database to store data permanently. Typically, Grid-monitoring tools involve small, local databases per service, populated by sensors or information providers with a certain regularity. Every update of such a database overwrites previously stored information, such that no service keeps long-term records of its own activity and status. Examples of monitoring systems integrated with Grids are:

- Globus Monitoring and Discovery Service (MDS) [5-16]: presently used by all the major Grid systems (EDG/LCG, NorduGrid/ARC, Grid3/OSG);
- Relational Grid Monitoring Architecture (R-GMA) [5-17]: used by EDG, gLite;
- GridICE [5-18]: used by parts of LCG, gLite, relies on MDS, R-GMA or other similar service;
- Monitoring Agents using a Large Integrated Services Architecture (MonALISA) [5-19]: used by Grid3/OSG.

The systems above are not application-specific, and are deliberately designed to accommodate only generic information. Nevertheless, each monitoring system can be set up to monitor different set of objects and their attributes.

In general, there are several objects that can be monitored:

- the computing facility (cluster, farm, pool)
    - the batch queue (if applicable)
        - a job in a queue
- the storage facility (disk array, tape storage)
- the file catalogue (data indexing service, such as RLS, Grid file system, etc)
- the VO user.

Each of these objects has a set of characteristic attributes that can be monitored. For example, a computing facility can be characterized by its architecture, number of processors, ownership, etc. A job can be characterized by its name, owner, resource consumption, location, submission host etc. Table 5-1 lists a typical set of such parameters as used/needed by ATLAS.

A functional monitoring system needs to collect information in real time from a number of sources, including our production database, the various Grid-information systems, and local batch systems. By now, each Grid flavour has well-developed monitoring tools and ways to browse the information, usually through web pages. Central effort on monitoring, covering not only job submission and distribution, but also file transfer activities, has so far suffered from insufficient manpower.

Job-monitoring tools that we developed for DC2 in 2004 have shown the limitations of the concept of the single Oracle-based production database; repeated monitoring queries could effectively slow down the response time much beyond expectations. One possible solution that is under discussion would consist in moving all records relative to completed jobs (or perhaps completed job blocks) either to different tables or to a different database instance, so that monitoring queries would have to deal only with a much smaller number of “active” records.

Table 5-1 Attributes to be monitored.

Computing facility and batch queue	Availability for submission Architecture and operating system Available software (run-time environment) Number of advertized CPUs Number of running and queued jobs
Job	Person submitting the job and his role Name of submission service Submission and destination sites Job type Time of submission Time of entry into a local queue Start and stop time Number of times the job was suspended and total time in suspended state Used resources: CPU time, memory, disk space Time in initialization and wait time to access input data Compute node characteristics Exit codes Initial estimates of CPU time and input and output data sizes Actual input and output data sizes
Storage facility	Total space on the Storage Element Free space on the Storage Element Type of interface Authorization scheme
File catalogue	Catalogue type Number of records
VO user	Jobs submitted by the user Jobs per role CPU consumption Storage consumption

Since all the monitoring systems are very different, ATLAS has not yet succeeded in producing a common monitoring tool for all three Grid flavours. Different approaches can be considered to achieve the goal of collaboration-wide monitoring:

- Create a monitoring interface to all the known information systems. The shortcoming of this approach is that it will not be able to monitor jobs at non-Grid sites, as those are not equipped with information providers.
- Design a monitoring client based on the information collected by the production database. The major limitation is that the production database contains no site-specific or user-specific information; also, it is updated at a comparatively slow rate.
- Instrument every job with an ATLAS-specific information-providing agent. Such an agent might not be able to collect complete information about the execution site or activities not related to the job, such as status of a data indexing service, but overall it appears to be the most appealing solution.

An important activity related to job monitoring is the identification of error conditions and of the subsequent actions to be taken, automatically or not. We have defined a set of “standard” (or rather, most frequent) error conditions and have asked the developers of all executors to make sure that the error codes that come back from each job step are passed back correctly and

mapped to our standard definition. In this way we hope to be able to increase the level of automation in the production system and concentrate on those job failures that may be due to ATLAS code.

Grid monitoring should in principle be done much more efficiently by the GOCs (Grid Operation Centres) than by the LHC experiments. In practice, because of the late start of the GOC activity (especially on the LCG Grid), we had to invest ATLAS effort into checking the status of the Grid sites, and the correctness of the information available through the information system, on a continuous basis. The set of tools for the LCG Grid provided recently by the LCG-GD group is very useful as a starting point and we are working with them on the customization and optimization of these tools for ATLAS.

5.4.7.4.2 Book-keeping

Specifics of book-keeping are:

- comprehensiveness: collected information must be as complete as possible and cover as many aspects as necessary;
- usability: the system must be able to deliver promptly any kind of stored information and return queries of arbitrary complexity;
- scalability: the system must be capable of storing and serving ever-increasing amounts of data.

Book-keeping procedures are inherently connected to monitoring. The difference is subtle from the client point of view, but essential from the server side: information collected about a process or an object must be preserved permanently, well after the object ceases to exist. A good example is the job-exit code: whenever a job finishes, it is removed from the execution site, hence it cannot be monitored any longer. In order to ensure that the job exit code can be retrieved after its end, the code has to be stored permanently, in a dedicated database. Clearly, local monitoring databases cannot offer such persistent storage service since it will negatively affect their performance. Moreover, storing book-keeping information in a widely distributed database requires non-trivial solutions to prevent data loss and to provide fast, reliable and properly authorized access.

A typical book-keeping task, often confused with monitoring, is that of obtaining a production-status report: the number of jobs submitted, finished, succeeded, failed, their categorization by failure reason, the integrated resource consumption, the contribution per site etc. Monitoring systems cannot address such tasks, as they do not store all these data persistently. The procedure of producing such a report is similar to monitoring: the view must be refreshed at a certain rate. However, each consecutive report must include data used in previous reports, which can be achieved only when the data are stored permanently.

Different Grid systems address book-keeping problems differently, but all known solutions involve a central database that stores job records. The difference comes in the technology used to populate the databases, and in the kind of information being stored. So far ATLAS has made no explicit use of the Grid-specific logging and book-keeping services. Instead, accounting and historical analysis relied on the records in the ATLAS production database, filled by the supervisors. This is an adequate approach, with only one major limitation: the production database stores only information about the jobs, hence it is impossible to estimate such parameters as the overall system load, efficiency, share of problematic sites, etc.

5.4.7.4.3 Accounting

By "accounting" we refer to the collection of data on the usage of CPU and storage resources by ATLAS jobs that are submitted through a central system. Other ATLAS jobs may be submitted by ATLAS users directly to the different Grids, but in that case we have no way of knowing what resources have been used.

Accounting has to be performed on two levels, the Grid side and the experiment's. Information must be available from the Grids on all jobs run by each user (or user group), and at the same time we must be able to retrieve easily such information from the production database. This information should allow cross-checks of accounting information.

The proposed separation of active from completed jobs in the production database will also help with the collection of accounting information, as statistics can be accumulated regularly on the completed jobs and stored in other database tables for future reference.

A set of tools to query the database to collect monitoring and accounting information in a uniform way is in preparation. Summary accounting information will be saved for further usage in a persistent structure. Tools for the easy retrieval of the information with web access are also being worked on.

5.4.7.5 Time-scales

After the end of the current round of productions for the Rome Physics Workshop (June 2005), there will be time to phase in the revised production system and test it during Summer 2005, before the next round of productions starts in late autumn 2005.

The next large-scale productions will see the generation of several tens of millions of events with a more realistic detector geometry (the "as-built" geometry), including misalignment and detector sagging, to be used for tests of trigger code and of the conditions database infrastructure, as well as of alignment and calibration procedures. This production activity is preliminary to the "Computing System Commissioning" operation in the first half of 2006, described in detail in Section 8.2.2.

5.5 Distributed Analysis on the Grid

The aim of the distributed-analysis project is to enable individual ATLAS users to use Grid resources for their analyses in physics and detector groups. Up to now the use of Grid resources on a large scale has been demonstrated only in the context of large-scale simulation and reconstruction of physics data. These activities are typically supported by small teams of Grid experts. The complications of computing on a global scale have been prohibitive so far for individual users and smaller groups, and they have not yet been able to take advantage of these resources.

Nevertheless, the computing needs after the start of LHC will require the use of Grid resources to support analysis on the required scale. In the following sections the ATLAS-specific aspects of distributed analysis will be presented in more detail. The requirements for data management and workload management for distributed analysis are discussed, and the various ATLAS activities are then presented in turn. Their current status is reviewed and future developments are outlined.

5.5.1 The ATLAS Distributed Analysis Model

Many aspects of Grid-based analysis have already been discussed in detail by several working groups [5-20]–[5-23]. These considerations are clearly relevant for the ATLAS activity; to complement these discussions, some ATLAS-specific aspects are presented. For the purpose of this discussion, we group together analysis tasks and any other medium-scale computing activity, such as individual testing of new simulation, calibration or reconstruction algorithms before putting them in production.

The need for distributed analysis follows from the distribution of the data in various computing facilities according to the ATLAS Computing Model, and by the availability of the CPU resources required to perform the actual analysis on large datasets. An analysis job will typically consist of a Python script that configures and runs a user algorithm in the Athena framework, reading from a POOL event file and writing a POOL file and/or ROOT histograms and/or n-tuples. More interactive analysis may be performed on large datasets stored as ROOT n-tuples. The work model of the different physics groups may vary within the ATLAS Collaboration, as it may be strongly influenced by the size of their relevant "signal" and "background" datasets; the distributed analysis system must be flexible enough to support all work models as they emerge.

Some important requirements on the distributed analysis system are the ease of use by collaboration members, the robustness of the system (including its interface to the Grid systems), and job traceability. In addition, the look and feel of the system should be the same whether one sends a job to one's own machine, a local interactive cluster, the local batch system, or the Grid.

5.5.2 Data Management

The ATLAS Data Management Project is responsible for the organization of the datasets at the various computing facilities according to the ATLAS Computing Model. A first model using local file catalogues and a global dataset location service has been outlined in Chapter 4. In this model a dataset may be distributed to a number of computing centres; a still open question is the integration of the dataset location service with the workload management system.

Users have to be provided with mechanisms for "random access" to relatively rare events out of a large number of files, in order to implement fast pre-filtering based on selected quantities (trigger bits and/or reconstructed quantities). In ATLAS a first iteration of the TAG datasets is in preparation and under initial tests. Selecting events from this TAG database, and then accessing the underlying event data base, will form an important verification of the capabilities of the data management system.

A further question to be studied is the storage of user data in the Grid environment. A large amount of data resulting from analysis jobs will have to be stored and managed. It is in the nature of the Grid that these datasets are not centrally known and require therefore an automatic management system. This implies the concept of file ownership and individual quota management based on this ownership. While many ideas have already been presented, practical implementations have yet to be demonstrated.

A final issue is the potential mismatch of user data management between the Grid environment and the local computer centres. Data that have been produced by analysis on the Grid will need further refinement by local interactive analysis. Transparent access to these data has to be provided, at least for data residing on storage elements close to the user.

Given the importance of data access to distributed analysis, there has to be a very close relation between the projects. Distributed analysis relies on the tools provided by the data management and has to verify that its requirements are taken into account. While many discussions have taken place and clear requirements for the interplay of the distributed data management system and distributed analysis have been identified, very little has been prototyped and tested so far. The second half of 2005 will see the implementation of the distributed data management system (described in Chapter 4) and the first tests of remote data access for user analysis tasks.

5.5.3 Workload Management

Assuming that the location of the data (or datasets) will determine the place where analysis is performed, it will be relatively simple to organize the overall analysis activities by placing the data according to the plans of the ATLAS Collaboration. This means that user jobs will be executed where a copy of the required data is already present at submission time. The problem is then just the normal fair-share of a batch system, which is by no means a trivial task if multiple users or working groups are active at the same time.

As pointed out in the HEPCAL2 document [5-22], there are several scenarios relevant for analysis:

- analysis with fast response time and high level of user influence;
- analysis with intermediate response time and influence;
- analysis with long response times and a low level of user influence.

As discussed in Ref. [5-22], it is the second scenario that is likely to be the most interesting and challenging. One can assume that this will cover most analysis activity.

Of particular interest are models that provide some sort of interactive response. In ATLAS the DIAL project [5-24] has been pioneering such fast response by low-latency batch queues. Some projects are studying approaches that go beyond the batch model, as in PROOF. A similar example is also the DIANE [5-25] model that is discussed below. It has to be pointed out that many aspects of such a model, such as resource sharing, have not yet been demonstrated in a realistic scenario.

5.5.4 Prototyping Activities

ATLAS users have not yet been provided with a common environment that gives access to ATLAS resources on the Grid. On the other hand, a number of prototyping activities have been developed within the ATLAS Collaboration, some of them in cooperation with national Grid partners; they are presented in turn in this section. Although none of these activities have provided ATLAS with a complete system that satisfies the requirements outlined above, they provide some of the building blocks with which we can construct the ATLAS distributed analysis system before the end of 2005.

5.5.4.1 AMI

The ATLAS Metadata Interface project provides metadata services to ATLAS (see Section 4.7.2). According to its design principles, AMI uses an RDBMS and provides access to metadata by a

Java API. A web service that provides for clients written in different programming languages has also been developed. For distributed analysis the user wants to select datasets based on some metadata, and retrieve the logical filenames forming the dataset. Recently new requirements for metadata have been defined [5-26]; an iterative process that includes interactions between the database developers and the distributed analysis community is necessary to improve the functionality of AMI for analysis applications.

5.5.4.2 DIAL

The DIAL project [5-24], supported by PPDG in the US, aims at providing a high-level service that simplifies access to the Grid for the user. The user submits a high-level task to the DIAL scheduler. The scheduler in turn takes care of splitting the job request and submitting the resulting jobs to the corresponding infrastructure, local batch or Grid. The server also provides additional services such as merging of result datasets, n-tuples and histograms. DIAL aims to be a comprehensive solution that provides access to Grid resources and to metadata catalogues from an integrated command line interface in ROOT and Python. A GUI is under development in close collaboration with the GANGA project. Recently the implementation has progressed significantly and essential features are now available; services submitting jobs to local LSF batch queues were put in place in BNL and at CERN in May 2005 and are currently under test.

5.5.4.3 Production System for Analysis

While it is possible to run analysis-type jobs through the production system, not all functionalities for supporting distributed analysis are currently available. At this point it is necessary that users run their own instance of supervisor and executor under their own control. Clearly there is also the need for development in the area of graphical and command line interfaces to hide some of the underlying complexity. On the other hand, the production system has been demonstrated to scale to several thousand jobs in parallel.

A related activity is the development of tools to control the production of the ATLAS combined test beam simulation and reconstruction; these tools have been evolved to include job monitoring and verification features. These developments are closely related and similar features will be available for Grid users.

5.5.4.4 GANGA

GANGA (Gaudi, Athena aNd Grid Alliance) is a joint project between ATLAS and LHCb [5-27], supported by GridPP in the UK. It is a user-centric interface for Athena job assembly, submission, monitoring, and control. GANGA is interfaced to various submission systems such as batch clusters (LSF, PBS), Grid infrastructures (LCG-2, gLite), and also experiment-specific production environments (DIRAC of LHCb). A variety of analysis, reconstruction, and simulation applications is supported by an extensible plug-in mechanism. An important aspect is the integration with the development environment that simplifies the definition of jobs. Additionally GANGA helps a user to keep track and history of jobs by saving input and output files in a systematic and coherent way. There is the possibility of running this job registry on a remote server, which enables users to monitor their jobs from various locations (desktops at work, laptops at home, etc.).

The upcoming release of GANGA 4 will be the basis for an extensive user evaluation in the ATLAS community.

5.5.4.5 DIANE

DIANE is a lightweight distributed framework for parallel scientific applications in a master–worker model [5-25]. It assumes that a job may be split into a number of independent tasks, which is a typical case for ATLAS analysis applications. The execution of a job is fully controlled by the framework which decides when and where the tasks are executed.

DIANE is based on a pull model: workers request tasks from the master. This approach is naturally self load-balancing. Additional reliability comes from the master which occasionally checks if workers are still alive to detect silent application crashes. DIANE workers act as daemons interacting with the master, which is a direct way of implementing interactive data analysis.

The current prototype of DIANE has proved useful for distributing ATLAS analysis tasks within a local cluster in the Swiss computing centre [5-28]. This functionality is important for the users and will likely continue to be needed also in the future. Distributed interactive analysis is simplified by working in the user's account and taking advantage of a shared filed system. It would be beneficial to have the local cluster functionality preserved and augmented. Work on the integration of this framework to Grid middleware is in progress; so far the possibility of running simulation jobs in parallel on distributed LCG-2 resources has been successfully tested. DIANE could complement batch-oriented systems by providing an interactive job execution with Grid resources on a large scale.

5.5.5 The ADA Project

ADA, the ATLAS Distributed Analysis system, has as a goal to enable ATLAS physicists to analyse the data produced by the detector and reconstructed by the production system, and to carry out production of moderately large samples. The system must perform well and be easy to use and access from the expected analysis environments (Python and ROOT) and flexible enough to adapt to other environments that might later be of interest.

Although ADA is being developed in the ATLAS context, the underlying software packages such as DIAL, GANGA and AMI have little dependence on other ATLAS software. Care is being taken to keep the entire system generic to allow use in other contexts that provide the required data and application wrappers.

The core of ADA consists currently of the DIAL services. Datasets can be imported from the AMI metadata database to the internal DIAL data catalogues so that they are readily and efficiently usable in the DIAL context. DIAL services can split jobs and submit them to local LSF batch systems at BNL and CERN; DIAL will also keep track of the jobs, collect the outputs, and make them available to interactive user sessions.

ADA will also provide its users with a graphical interface to aid in examining data and specifying, submitting and monitoring jobs. This work is being done by the GANGA project, making use of the Python binding to DIAL.

ADA had its first public release in April 2005. There are two analysis service instances running at BNL (one for long-running jobs and one that provides interactive response) and one at CERN. ADA has clients to enable use of the AMI service but so far little of its data is resident there. All of the reconstructed data from the first ATLAS data challenge and part of the datasets produced since are available as ADA datasets; work is in progress to complete the catalogues with all existing ATLAS data. Transformations have been defined to allow users to access the data and insert their own analysis algorithms.

ROOT and Python clients are available providing the full DIAL functionality as described earlier. These and the above analysis services, plus the AOD analysis and the combined n-tuple transformation, are the bulk of the functionality available in the first ADA release.

As of June 2005, we are in the process of re-evaluating the experience we gained so far with the development of the DIAL-based ADA prototype. Several related projects, described above, have to be brought together in a coherent way so that the end-users will not have to change work model, or environment, depending on whether they want to run on a local machine, a local interactive or batch cluster, or the Grid.

A large amount of simulated data, produced for the June 2005 Rome Physics Workshop, and of real data, produced by the 2004 combined test beam, is distributed around the world and is now potentially accessible using Grid tools. These datasets will be used during 2005 to test the different options outlined above, and to define the baseline solution to be developed further.

5.6 Testing and Commissioning of the System

By the end of 2005 we must have a working production and analysis system that can scale to the needs of the ATLAS user community in 2007. This system will be used for the Computing System Commissioning activities that are planned to start in early 2006 (see Section 8.2.2). In order to be ready in time, we have established a list of high-level milestones for the development of the production and analysis systems and related activities:

- Mid-July 2005: decision on the future developments of the distributed analysis system.
- End July 2005: major re-work of the components of the production system.
- End July 2005: report on the first tests of gLite middleware components.
- End September 2005: new version of the production system fully tested and deployed.
- Mid-October 2005: start of productions of simulated events matching the cosmic-ray data collected by the ATLAS detector.
- Mid-November 2005: start of large-scale productions to generate simulated events for Computing System Commissioning in early 2006.

5.7 References

5-1 *LCG Technical Design Report*, CERN-LHCC-2005-024

5-2 VOMS:
https://edms.cern.ch/file/571991/1/voms-guide.pdf

5-3 G. Carcassi *et al.*, *A Scalable Grid User Management System for Large Virtual Organization*, proceedings of CHEP04, Interlaken (Switzerland), 2004. See also
http://osg.ivdgl.org/twiki/bin/view/Integration/GumsAdmins

5-4 M. Branco, *DonQuijote – Data Management for the ATLAS Automatic Production System*, proceedings of CHEP04, Interlaken (Switzerland), 2004

5-5 Windmill project:
http://heppc12.uta.edu/windmill/

5-6 R. Sturrock *et al.*, *Performance of the NorduGrid ARC and the Dulcinea Executor in ATLAS Data Challenge 2*, proceedings of CHEP04, Interlaken (Switzerland), 2004

5-7 M. Mambelli *et al.*, *ATLAS Data Challenge Production on Grid3*, proceedings of CHEP04, Interlaken (Switzerland), 2004

5-8 D. Rebatto, *The LCG-2 Executor for the ATLAS DC2 Production System*, proceedings of CHEP04, Interlaken (Switzerland), 2004

5-9 J. A. Kennedy, *The role of legacy services within ATLAS DC2*, proceedings of CHEP04, Interlaken (Switzerland), 2004

5-10 A. Chervenak, E. Deelman, I. Foster *et al.*, *Giggle: A Framework for Constructing Scalable Replica Location Services*, proceedings of SuperComputing 2002, Baltimore (USA), 2002

5-11 J. Frey, T. Tannenbaum, I. Foster, M. Livny and S. Tuecke, *Condor-G: A Computation Management Agent for Multi-Institutional Grids*, *HPDC10*, San Francisco (USA), 2001,
http://www.cs.wisc.edu/condor/condorg

5-12 Grid Production System review report:
https://uimon.cern.ch/twiki/bin/view/Atlas/GridProdToolsFinalReport

5-13 D. Barberis *et al.*, *Report of the ATLAS Grid Tools Task Force*, ATL-SOFT-INT-2005-004, April 2005

5-14 Eowyn project:
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/DC/PRODSYS/Eowyn-v0.3.pdf

5-15 A. Tsaredgorotsev *et al.*, *Dirac – The distributed MC production and analysis for LHCb*, proceedings of CHEP04, Interlaken (Switzerland), 2004

5-16 Globus Monitoring and Discovery Service (MDS):
http://www.globus.org/toolkit/mds

5-17 Relational Grid Monitoring Architecture (R-GMA):
http://www.r-gma.org

5-18 GridICE:
http://infnforge.cnaf.infn.it/gridice

5-19 Monitoring Agents using a Large Integrated Services Architecture (MonALISA):
http://monalisa.cacr.caltech.edu

5-20 P. Buncic *et al.*, *Architectural Roadmap towards Distributed Analysis*, CERN-LCG-2003-33, October 2003

5-21 F. Carminati *et al.*, *Common Use Cases for a HEP Common Application Layer – HEPCAL*, LCG-SC2-2002-20, October 2002, revised in November 2004

5-22 D. Barberis *et al.*, *Common Use Cases for a HEP Common Application Layer for Analysis – HEPCAL II*, CERN-LCG-SC2-2003-32, October 2003

5-23 D. Olson and J. Perl, *Grid Service Requirements for Interactive Analysis*, PPDG CS11 Report, September 2002

5-24 DIAL project:
http://www.usatlas.bnl.gov/~dladams/dial

5-25 J. Mosciki, *Distributed Computing and Oncological Radiotherapy: Technology Transfer from HEP and Experience with Prototype Systems*, proceedings of CHEP04, Interlaken (Switzerland), 2004. See also
http://cern.ch/diane

5-26 S. Hanlon *et al.*, *Unlucky for Some: The Thirteen Core Use Cases for HEP Metadata*, December 2004, GridPP Metadata Working Group

5-27 GANGA project:
http://ganga.web.cern.ch

5-28 S. Gadomski *et al.*, *The Swiss ATLAS Computing Prototype*, ATL-SOFT-PUB-2005-003

6 Computing Operations

6.1 Introduction

In this chapter we describe the main computing operations that ATLAS will have to run. We begin with the preparation and the distribution of the ATLAS software. We then review the operations which will be run on Tier-0, Tier-1 and Tier-2. We explain how we intend to use the ATLAS Virtual Organization and how we intend to operate the productions.

Over the past few years the validation of the Computing Model has been done by running data challenges of increasing scope and magnitude, as was proposed by the LHC Computing Review [6-1]. We describe briefly the two data challenges run since 2002 and how massive productions have been performed in order to provide simulated data to the physicists and to reconstruct and analyse real data coming from test-beam activities.

6.2 Software Distribution and Deployment

6.2.1 Preparation and Distribution of ATLAS Software

As described in Section 3.14, ATLAS software builds are organized in multiple domains:

- Nightly builds;
- Developer releases;
- Production releases;
- Bug-fixes or patches to the releases.

What is relevant for the Computing Operation are the production releases, which hold the stable code, used for production and end-user analysis, and normally occur every 4–6 months. The build processes are supervised by the Release Coordinator; it is his responsibility to ensure that all software components, including those not produced by ATLAS (external packages), can work together in a coherent way.

The physical builds and patching of the releases and the software distribution kits are supervised by the Librarian. He takes care of the installation and validation of the software at CERN and of the preparation of the Distribution Kit [6-2].

After sufficient testing, the kit is distributed to the external sites where the Validation Kit has to be successfully run. If problems occur in the full procedure, patches could be applied and the full distribution and validation be rerun until the validation tests are satisfactory.

6.2.2 ATLAS Software Deployment on the Grid

ATLAS software is installed on the Grid in different ways, depending on the Grid flavour. These different ways are mostly due to the fact that Grid sites are heterogeneous, using different oper-

ating systems and following different installation policies. It should be noted that the Grid sites are not necessarily High Energy Physics oriented sites.

Usually the installation of a new version is initiated by one of the Software Group Managers for a particular Grid. For some sites the installation is performed by the site administrator.

6.2.2.1 LCG Grid

The SIT distribution kit is used for the deployment of ATLAS software in LCG Grid sites.

The installation process is executed via special grid jobs, using the standard LCG tools, customized for ATLAS-specific needs. An installation job, when landing on a target machine of a site, downloads and installs the requested release in an area that is shared at the cluster level. This area, the Experiment Software Area, is pointed to at run time by the environment variable VO_ATLAS_SW_DIR. The main directory of the release ($SITEROOT) is set to $VO_ATLAS_SW_DIR/software/<rel_num>, where <rel_num> is the "dots and numbers" release tag (for example "10.0.1"). The standard procedure also installs the correct compiler for the selected release, along with the other binaries in the Experiment Software Area. The software installation is performed using Pacman [6-3].

After the installation step, the software is validated via the Kit Validation tool. If the validation step is also successful, then a tag, used to uniquely identify the installed release, is published into the Information System of the local Computing Element, so that it can be used during the match-making step for the submissions of the jobs. Similar jobs are used to perform the removal of an installed release and to remove the release tags from a site.

The information about the installation operations and status are kept at central level in a MySQL database. A web page for presenting the installation information is also available [6-4].

6.2.2.2 NorduGrid

The Nordic Computing Grid (NorduGrid [6-5]) is built of many different systems (RedHat, SuSe, Debian, etc.). A full rebuild of the ATLAS software is generally required by NorduGrid sites, in order to make it compatible in native mode with the different architectures. The recompiled code is then packaged as RPMs and distributed to the sites. The installation is performed by the site administrator, as no Nordic site allows non-root users to write into the disk areas used for the run-time libraries of the experiment software and to publish the tags needed to identify the installed release in the Information System.

After the installation step the software is validated in local mode through a Kit Validation session. If this step is successful, then a further check, done by executing another Kit Validation through a grid job, is performed. This second check ensures that the release tag is also published correctly and that the script, used for setting up the runtime environment, is working properly.

All ATLAS software used in NorduGrid, up to release 10.0.1, has been fully recompiled from scratch, including the external packages. However, starting from release 10.0.1 the SIT Pacman distribution has been used, owing to the limited manpower available to perform the full rebuild. This limited the number of available sites to <6 sites, since the ~14 remaining sites cannot use the Pacman installations because of management reasons. Pacman, in fact, does not produce any record in the system databases, and this is not accepted by all sites.

6.2.2.3 Grid3/OSG

The installation of ATLAS software on US Grid3/OSG sites is done using Pacman, and built upon the Grid3/OSG Information Service infrastructure. The SIT distribution kit is used for software deployment.

The remote install scripts query the sites, to find the corresponding remote locations, then transfer the installation scripts and perform a local Pacman installation. At the end of the process the ATLAS application information is published and/or updated in the site information services and the installation web page.

6.2.2.4 Future of Software Installation on the Grid

The Software Infrastructure Team (SIT) is working on a new standardized installation procedure which should be identical for Grid and non-Grid sites. The new installation kit will allow both to install the compiled libraries and to perform a full cycle of compilation, creation of libraries and installation. The latter way to proceed is necessary for sites running unusual operating systems.

As mentioned earlier, most of the time the installation of a new version of the software is initiated by the Software Group Manager by sending jobs on the Grid. However, in some cases, especially when the compilation of the source code is required, the installation is initiated by the local ATLAS site administrator.

The new installation kit will trigger a validation operation which will consist in running several applications producing histograms which will be compared with reference ones. Depending on the results of the comparison, the installation will be considered as validated or not, in both cases the person responsible for the installation will be notified.

6.3 Production Operations

6.3.1 Operations at Tier-0

As stated in the Computing Model, the Tier-0 facility at CERN is responsible for the following operations:

- Calibration and alignment;
- First-pass ESD production;
- First-pass AOD production;
- TAG production;
- Archiving of primary RAW and first-pass ESD, AOD and TAG data;
- Distribution of primary RAW and first-pass ESD, AOD and TAG data.

6.3.1.1 General Organization

Figure 6-1 shows a diagram of the proposed high-level organization of the Tier-0. RAW data enters the Tier-0 from the Event Filter (EF) at a rate of 320 MB/s. This data consists of four streams: the main physics stream and the smaller streams of calibration/alignment events (45 MB/s), the express stream (~6 MB/s), and the much smaller stream of pathological events. RAW, ESD,

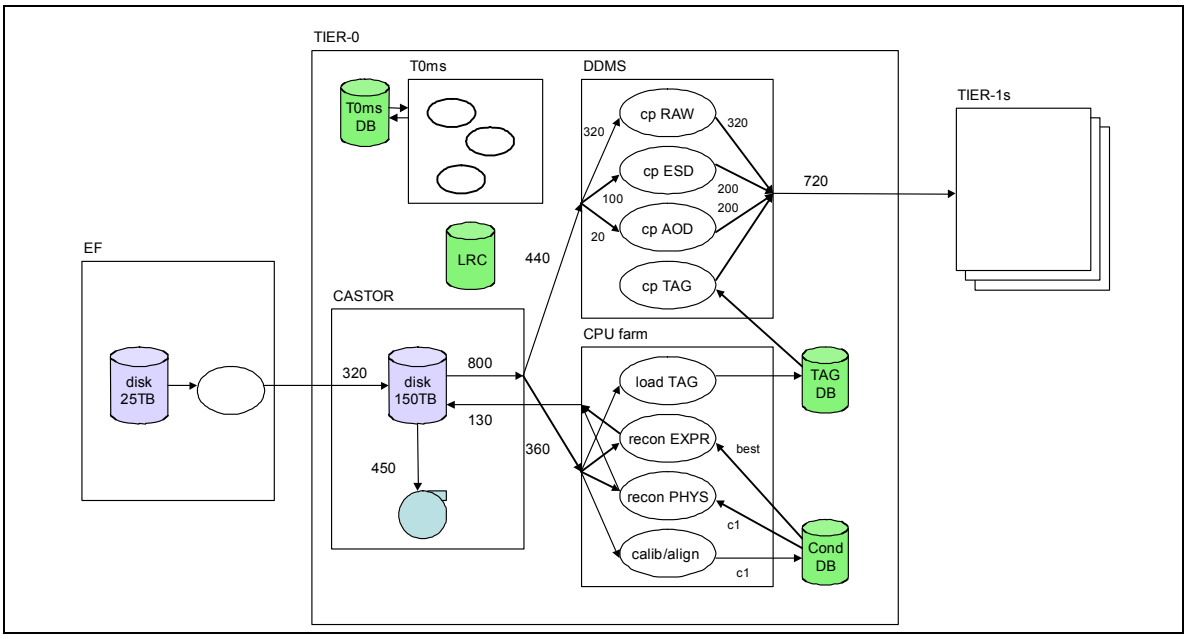

Figure 6-1 Tier-0 organization and data transfer rates between components (Numbers are in units of MB/s).

AOD and TAG data are copied out of the Tier-0 to the Tier-1s at a rate of ~720 MB/s. The Tier-0 itself consists of several components:

- Castor mass storage system and local replica catalogue;
- CPU farm;
- Conditions DB;
- TAG DB;
- Tier-0 management system and Tier-0 production database;
- Data management system.

The main storage facility is the Castor hierarchical mass storage system. This system will provide the necessary disk space for both the input buffer (125 TB, corresponding to 5 days of data-taking) and the output buffer (25 TB, or 2 days). A process running in the EF system will transfer the RAW files from the EF output buffer into this Castor component. The first-pass reconstruction processes will read their RAW input and write their ESD, AOD and TAG output onto this disk space. The Castor migrator will copy all files to tape as soon as possible with an average writing speed of 440 MB/s (320 RAW + 100 ESD + 20 AOD + 0.2 TAG).

Unless the processes reading the Castor data are stalled for a very long time (several days), all files will be used before being purged from the disks and recall from tape will not be necessary.

In the very unlikely case that such a big backlog is built up, care will be taken that processing it does not interfere with real-time processing, ensuring that no new backlog is created.

The main computing resource is the Tier-0 reconstruction farm. All jobs processing the physics and the express RAW data files into ESD, AOD and TAG data files run on this farm. In addition, the jobs processing the calibration and alignment RAW stream into conditions records in the conditions DB and the jobs uploading the TAG files into the TAG DB will be run on this CPU farm.

The distribution of the Tier-0 data products to the Tier-1s will be handled by a dedicated instance of the ATLAS distributed data management system (DDMS). The DDMS will provide the service accepting file movement requests and ensuring that these requests are eventually executed. The DDMS interacts with the Castor component through its gridftp interface and expects a local replica catalogue to be duly filled with entries for all the files stored on the Castor system.

The final component is the Tier-0 management system (T0MS) that will orchestrate the Tier-0 operation. This system has to define and execute jobs on the reconstruction farm and fill the DDMS with transfer requests. In that respect it is very similar to the ATLAS production system and indeed current expectations are that it will be built upon this production system.

There is one important difference: Tier-0 operation is largely data-driven while the production system is largely job-driven. Additional logic will be needed to translate the events of file creations into the appropriate job definitions.

Not all runs will be of equal length and not all files will be filled up to the same amount of events. This poses little complication for the jobs processing a single input file into a single output file, but it will require some attention for the jobs processing multiple input files.

Splitting the AOD generation from the ESD generation allows one to produce fewer and larger AOD/TAG files, but at the expense of reading the input ESD again from disk, thus adding an additional 100 MB/s to the reading rate of the Castor component.

Alternatively one could produce ESD and AOD in one job and merge the resulting small AOD files instead. This scheme naturally allows merging the different AOD channels with different group sizes compensating for any (very likely) imbalance between the channels. The intermediate small AOD files could then be discarded. This scenario would only create an additional 20 MB/s reading load on the Castor component.

It is expected that initially, and probably always, a human intervention will be needed to verify the results of the calibration and alignment processing. No physics RAW processing should start until after this “green light”. The physical implementation of this “green light” can be the act of tagging the data in the conditions database.

Given that the Tier-0 will be data-driven, the question arises of how data appearance will trigger actions. The simplest and most robust solution is probably to just periodically check the catalogue(s) for new data. In case, for some reason, this scheme is too naive, one could envisage a simple messaging system based on a message database.

The state of the T0MS is persistified in the Tier-0 production database and will probably be an extension of the state of the ATLAS production system. In the event of a T0MS crash, a restart from the T0MS DB should be possible with close to zero loss of information

6.3.1.2 Fault Tolerance

Scenario 1: No or degraded bandwidth on EF–Tier-0 link

On the EF side this means that data will enter the EF output buffer faster than they can be moved to Castor. A backlog will build up in the EF output buffer at a rate of (1 – efficiency) x 320 MB/s. Eventually the EF output buffer will become full and data will be lost. Because the expected average data taking time per day is only 50k seconds (~60%), we expect that even a one-day backlog can be compensated for within the order of one week of operation.

Scenario 2: No or degraded bandwidth from/to Castor

For the distribution process to the Tier-1s this means that a backlog will build up inside Castor. Data will not be lost but may eventually be purged from disk and hence need an expensive tape recall when needed later on. For the reconstruction and calibration jobs running on the CPU farm, the implications are that staging in and out of files will slow down and consequently the overall job rate will decrease. Note that this slowdown causes a corresponding slowdown of output file creation and hence will certainly not aggravate the situation.

Scenario 3: No or degraded bandwidth to tape within Castor

Eventually (after 5/(1 – efficiency) days) all files on disk will be in need of migration to tape. Two possibilities exist now. Strategy one will purge the files from disk anyway to make room for the new arriving files. Most likely, most files in the mean time will have already been distributed to the Tier-1 centres and hence could be recovered from there. So, this strategy requires some minimal coordination between the purging process and the distribution process to ensure that files that are indeed replicated out already are purged first. In addition, an asynchronous recovery process could attempt to recover the lost primary tape copies from the Tier-1 centres when spare bandwidth is available. Strategy two will not purge non-migrated files from disk and hence will stop input into Castor. The consequences of this are explained above.

Scenario 4: No or degraded CPU capacity in the CPU farm

The consequences of this are that RAW data enter the Tier-0 faster than they can be processed. The Castor disk will fill up with unprocessed RAW data files and eventually (after 5/ (1 – efficiency) days) RAW files will be purged from disk. Processing these files later on, when the CPU capacity is restored or the RAW inflow has decreased sufficiently, will require an expensive recall from tape. Care should be taken that the processing of these purged RAW files does not decrease the overall CPU farm efficiency to the point that more unprocessed RAW files are purged then recovered. A zero-th order strategy ensuring this, is to process files that need tape recall only when no other files can be processed.

Scenario 5: No or degraded bandwidth to the Tier-1 centres

The consequences of this are that data (RAW, ESD, AOD) enter Castor faster than they can be distributed to the Tier-1s. The Castor disk will fill up with undistributed data files and eventually (after 5/ (1 – efficiency) days) files will be purged from disk. Distributing these files later on, when the bandwidth is restored, will require an expensive recall from tape. Care should be taken that the recall of these purged files does not decrease the overall Castor efficiency to the point that more problems are caused than solved. Tape recall could, for example, destructively interfere with the migration to tape.

Scenario 6: No or degraded bandwidth to some Tier-1 centres

Here we need to distinguish between the three different distribution strategies for RAW, ESD and AOD.

For RAW, on average, one tenth of the data will be distributed to one of the 10 Tier-1s. In case of bandwidth problems to this Tier-1, one can either decide to distribute it to another Tier-1 instead, or stall the distribution until the bandwidth is restored. Strategy one requires an additional component monitoring the distribution and dynamically redirecting the scheduled transfers. Strategy two has the same consequences with respect to tape recalls as the previous scenario.

ESD needs to be distributed to two Tier-1 centres, so it is in fact equivalent to two times the RAW situation.

AOD needs to be replicated to all Tier-1 centres, so a strategy redirecting the copy cannot be applied.

Given that the Tier-1 centres are supposed to have a very high level of availability, current thinking is that it will be sufficient to simply stall the transfers and not enable dynamic redirection.

Scenario 7: No or degraded response of TAG database

As the loading of the TAG database from the TAG files is asynchronous with respect to the other processing anyway, in the worst case the TAG files could be purged from disk before they are used. Given the huge size of the Castor buffer and the small size of the TAG files, even the latter seems close to impossible. A little bit of care must be taken so that not all or even a small fraction of CPUs are occupied with TAG loading jobs waiting for the TAG database.

Scenario 8: No or degraded response of Conditions DB

No or degraded Conditions DB response will stop, crash, or slow down all jobs running on the CPU farm, and hence have the same effect as scenario 5. In this respect it would be advantageous for reconstruction jobs to access the conditions database in a concentrated way at start-up rather than continuously; in any case, all events in a single RAW data file are produced within at most a few minutes and with the same trigger conditions.

Scenario 9: Crash of Tier-0 management system

A crash of the Tier-0 management system will cause no new actions to be initiated. Most importantly, no new jobs will be submitted to the CPU farm. With an average job length of several hours, a simple (automatic) reboot within O(10) minutes would probably already suffice. In addition, like the multiple supervisor set-up in the ATLAS production system, one could envisage running multiple collaborating instances of the T0MS.

6.3.1.3 Calibration and Alignment

Calibration and alignment processing refers to the processes that generate “non-event” data that are needed for the reconstruction of ATLAS event data. These non-event data are generally produced by processing some raw data from one or more sub-detectors, rather than being raw data themselves. The input raw data can be in the event stream (either normal physics events or special calibration triggers) or can be processed directly in the sub-detector readout systems. The output calibration and alignment data will be stored in the conditions database.

At the moment it has not yet been decided whether all the calibration processing will run as a single job or be split in the event dimension and/or per detector. In any case it is certain that at least some results will depend upon data spread over the complete run, and hence cannot be computed until the complete run is available. In that respect, it may be of limited usefulness to introduce the complication of starting part of the calibration processing before a run is completed.

Calibration and alignment activities will also occur elsewhere, notably at the CERN Analysis Facility. These will be different, being more developmental in nature, intensive in human effort and not automatic. The time-scales involved will typically be longer, and will influence the latency of the reprocessing of the data, not the first processing at the Tier-0.

6.3.1.4 First-Pass ESD Production

First-pass ESD production takes place at the Tier-0 centre. The unit of ESD production is the run, as defined by the ATLAS data acquisition system. ESD production begins as soon as RAW data files and appropriate calibration and conditions data are available. The Tier-0 centre provides processing resources sufficient to reconstruct events at the rate at which they arrive from the Event Filter. These resources are dedicated to ATLAS event reconstruction during periods of data taking. The current estimate of the CPU power required is 3000 kSI2k (approximately 15 kSI2k-seconds per event times 200 events/second).

Note that this is the capacity needed to keep up with the RAW data in real time. It is expected that on average only 50k seconds per day data will effectively be taken, so on average only ~60% of this capacity is needed. However, as data-taking periods may be distributed unevenly and as occasionally backlogs will need to be caught up with, the remaining 40% has to be kept as a safety margin.

A new job is launched for each RAW data file arriving at the Tier-0 centre. Each ESD production job takes a single RAW event data file in byte-stream format as input and produces a single file of reconstructed events in a POOL ROOT file as output. With the current projection of 500 kB per event in the Event Summary Data (ESD), a 2-GB input file of 1250 1.6-MB RAW events yields a 625-MB file of ESD events as output.

6.3.1.5 First-Pass AOD Production

Production of Analysis Object Data (AOD) from ESD is a lightweight process in terms of CPU resources, extracting and deriving physics information from the bulk output of reconstruction (ESD) for use in analysis. Current estimates propose a total AOD size of 100 kilobytes per event.

As AOD events will be read many times more often than ESD and RAW data, and analysis access patterns will usually read all AOD of all events belonging to this or that physics channel, it makes sense to physically group the AOD events into streams reflecting these access patterns. Of course, there are many physics channels and many sensible groupings. Satisfying all of them would lead to an excessive number of streams and most likely duplication of events across streams.

As a compromise we expect the physics community to define of the order of 10 mutually exclusive and maximally balanced streams. All streams produced in first-pass reconstruction share the same definition of AOD. Streams should be thought of as heuristically-based data access optimizations: the idea is to try to reduce the number of files that need to be touched in an average

analysis, not to produce perfect samples for every analysis. Note that we expect the actual definition of the 10 streams to evolve over time.

The Computing Model foresees a separation of the AOD production from the ESD production in order to avoid an excessive number of small (10 MB and less) AOD files. A separate AOD building step would read in 50 ESD files and produce 50 times fewer and 50 times bigger AOD files.

The disadvantage of this model is that the ESD data needs re-reading from Castor adding an extra 100 MB/s on the reading bandwidth. Additionally, it does not naturally allow the grouping of AOD streams with even fewer events into bigger groups (i.e. the stream imbalance is preserved).

An alternative strategy would be to produce the 10 AOD files (and the TAG file) per RAW file together with the ESD file in a chained job. The resulting small AOD files could then be merged together per AOD channel, using a per-channel optimal grouping size in a separate merging job. As the total AOD size is only 100 kB, this would cause an increase of only 20 MB/s on the Castor reading bandwidth. The total number of jobs seen by the CPU farm remains the same.

6.3.1.6 TAG Production

AOD production jobs simultaneously write event-level metadata (event TAGs), along with "pointers" to POOL file-resident event data, for later import into relational databases. The purpose of such event collections is to support event selection (both within and across streams), and later direct navigation to exactly those events that satisfy a predicate (e.g., a cut) on TAG attributes.

To avoid concurrency control issues during first-pass reconstruction, TAGs are written by AOD production jobs to files as POOL "explicit collections," rather than directly to a relational database. These file-resident collections from the many AOD production jobs involved in first-pass run reconstruction are subsequently concatenated and imported into relational tables that may be indexed to support query processing in less than linear time.

While each event is written to exactly one AOD stream, references to the event and corresponding event-level metadata may be written to more than one TAG collection. In this manner, physics working groups may, for example, build collections of references to events corresponding to (possibly overlapping) samples of interest, without writing event data multiple times. A master collection ("all events") will serve as a global TAG database. Collections corresponding to the AOD streams, but also collections that span stream boundaries, will be built during first-pass reconstruction.

6.3.1.7 Archiving of Primary RAW and First-Pass ESD, AOD and TAG Data

RAW data and first-pass ESD, AOD and TAG data will be permanently archived at the Tier-0 with a copy on the Tier1s. For 2008 we estimated (Chapter 7) that we will need 6.2 PBof tape storage at the Tier-0 and of the order of 9.0 PB for all the Tier-1s.

6.3.1.8 Distribution of Primary RAW and First-Pass ESD, AOD and TAG Data

ATLAS decided to use a Distributed Data Management system and has started the development of the final version of that system called DonQuijote-2 (DQ2). Based on the experience gained while running the Data Challenges, the system is fully described in Section 4.6; in particular the system has been designed in such a way that a multiplicity of Grids can be supported.

DQ2 will be used for all types of data transfer (RAW, ESD, AOD, TAG, Monte Carlo, etc.), for all possible exchanges between Tiers in any direction. We expect to test the full system in the Service Challenge 3 in Autumn 2005 and have it ready for the computing commissioning scalability tests in early 2006.

6.3.2 Operations at Tier-1s

As described in the Computing Model, ATLAS Tier-1s will be a set of full-service computing centres for the Collaboration. They will provide a large fraction of the raw and simulation data storage for the experiment. They will also provide the necessary resources to perform reprocessing and analysis of the data.

In accepting to host a copy of the raw data, each Tier-1 also accepts to provide access to this data for the entire collaboration and to provide the computing resources for the reprocessing. It is not expected that access to all of the hosted raw data should be available on a short latency but at least a reasonable fraction of it should be on fast disk storage for calibration and algorithm development. However, access to ESD, AOD and TAG datasets should always be available with short latency (on 'disk'), at least for the most recent version of processing, while previous versions will be available with a longer latency (on 'tape').

In accepting data from Tier-2, a Tier-1 accepts to store them in a permanent and safe way and to provide access to it in agreement with current ATLAS policy. This is true for both simulated and derived data.

In order to meet these goals a set of services should be provided:

- Mass storage data archive. This service should be reliable enough to accept a copy of the raw data.
- Infrastructure for site security and access restrictions.
- Prioritization of access to data and processing resources in agreement with ATLAS policy.
- Accounting information for both processing and data storage.
- Database services to allow replication and caching of database information for calibration data, parameter set of data, data set book-keeping data, etc. in agreement with the ATLAS DDM strategy.
- Publication of the necessary information to be used by the Grid services.

These services should be of high quality in terms of availability and response time.

Each local site will be primarily responsible for its hardware, storage and network services. This will be organized by the Tier-1 board.

The operations at the Tier-1 will be the responsibility of the Computing Operations Group. The production managers of the working groups will be allowed to submit their respective productions in agreement with the general ATLAS production policy.

Full read–write access to Tier-1 facilities will be restricted to the Production Managers of the working groups and to the central production group for reprocessing and overall data management.

It is not excluded that a given Tier-1 provides services to local users beyond its ATLAS Tier-1 role.

6.3.3 Operations at Tier-2s

Tier-2 facilities will play a range of roles in ATLAS such as providing calibration constants, simulation and analysis. They will provide a flexible resource with large processing power but with a more limited availability compared to a Tier-1 in terms of mass storage accessibility and network connectivity.

The basic functions include:

- Fast and detailed Monte Carlo event generation, simulation and reconstruction.
- Data processing for physics analysis, generally this will be done in a chaotic way.
- Data processing for calibration and alignment tasks, code development and detector studies.

In order to meet these requirements, a set of services should be provided. They include:

- Medium- or long-term storage for required samples. For analysis work these will be mostly AOD but could include some fraction of ESD or RAW data for more detailed studies.
- Transfer, buffering and short-term caching of relevant data from Tier-1 and transfer of produced data to Tier-1 for storage.
- Provision and management of local temporary working space for analysis, calibration of development tasks.
- Database services in agreement with the ATLAS DDM strategy.
- Prioritization of access to data and computing resources.
- Accounting for processing and data storage.
- Publication of the necessary information to be used by the Grid services and the ATLAS Production System.

The simulation production will be the responsibility of the production managers. Analysis or detector studies activities will be the responsibility of the corresponding working groups which should appoint a production manager.

Read access should be provided to all members of the ATLAS VO. Write access should be available to ATLAS Production Managers.

A Tier-2 will have the possibility to request that some specific data should be replicated to it, for example AOD data of interest to the physics community it serves. The request will be sent to the Distributed Management System which will trigger the necessary operations.

6.3.4 Production and Databases

In the ATLAS Computing Model, various applications require access to the data resident in relational databases. Examples of these are databases for detector production, detector installation, survey data, detector geometry, online run book-keeping, run conditions, online and offline calibrations and alignments, offline processing configuration and book-keeping.

The Database Project is responsible for ensuring the integration and operation of the full distributed database and data management infrastructure of ATLAS. The Distributed Database Services area of the Project is responsible for the design, implementation, integration, validation, operation and monitoring of all offline database services.

ATLAS data processing will make use of a distributed infrastructure of relational databases in order to access various types of non-event data and event metadata. Distribution, replication, and synchronization of these databases, which are likely to employ more than one database technology, must be supported according to the needs of the various database service client applications. In accordance with the LCG deployment model, we foresee Oracle services deployment in the Tier-0 and Tier-1 centres and MySQL in smaller settings (Tier-2, Tier-3, Tier-4) and for smaller applications.

To achieve the integration goal, ATLAS is developing the Distributed Database Services client library that serves as a unique layer for enforcing policies, following rules, establishing best practices and encoding logic to deliver efficient, secure and reliable database connectivity to applications in a heterogeneous, distributed-database services environment.

ATLAS favours common LHC-wide solutions and is working with the LCG and the other experiments to define and implement common solutions. ATLAS initiated and is actively involved in the LCG distributed database deployment project. ATLAS also initiated and is contributing manpower for the integration of the Distributed Database Services client library functionalities in the Relational Access Layer of the LHC-wide project POOL.

Experience from the deployment of grid technologies in the production environment for ATLAS Data Challenges demonstrated that a naïve view of the grid as a simple peer-to-peer system is inadequate. Our operational experience shows that a single database on top is not enough to effectively run production on a federation of computational grids. Going beyond the existing infrastructure, an emerging hyperinfrastructure of databases complements the Tier-0–Tier-1–...–Tier-N hierarchy of computational grids. The database hyperinfrastructure separates concerns of the local environments of applications running on worker nodes and the global environment of a federation of computational grids. Databases play a dual role: both as a built-in part of the middleware (monitoring, catalogues, etc.) and as a part of the distributed workload management system necessary to run the scatter–gather data processing tasks on the grid. It is the latter role that supplies operations history that is providing data for important 'post-mortem' analysis and improvements.

6.3.5 Monitoring and Book-Keeping

ATLAS has not yet produced a common monitoring tool for all three Grid flavours. Different approaches can be considered to achieve the goal of collaboration-wide monitoring:

- Create a monitoring interface to all the known information systems. The shortcoming of this approach is that it will not be able to monitor jobs at non-Grid sites, as those are not equipped with information providers.
- Design a monitoring client based on the information collected by the production database. The major limitation is that the production database contains no site-specific or user-specific information; also, it is updated at a comparatively slow rate.
- Instrument every job with an ATLAS-specific information-providing agent. Such an agent might not be able to collect complete information about the execution site or activities not related to the job, such as status of a data indexing service, but overall it appears to be the most appealing solution.

Very important information is contained in job exit code. ATLAS Data Challenges demonstrated that it is essential for jobs to return an exit code of high granularity that would simplify production management, facilitate debugging process, and generally help in producing a clear overview of the data processing. On some Grid systems, jobs can get lost, producing no exit code whatsoever, or any other information. A special code range is assigned to such jobs in the book-keeping system. Error reporting can have a finer structure, by also adding error-type code to the information being collected.

Presently, no Grid offers a book-keeping system that is adequate for ATLAS needs. It should be a primary goal of the LCG project to provide a common view on monitoring across the member deployments. In the long term, placing the load for this on the experiment is unsustainable. The solution provided should allow a flexible schema so that the production and individual users can instrument their jobs. Moreover, there are no accepted standards as to what has to be stored as a persistent usage record. The GGF Usage Record Working Group is developing an implementation-independent schema for the resource usage record, suitable for book-keeping purposes. However, even if all the Grids were to adopt this format, it will cover only the most generic information, likely to be insufficient for ATLAS needs. Therefore, information stored in an ATLAS-specific database will remain essential, and it is important to ensure that all the necessary data are collected, reliably stored, and can be easily retrieved.

6.4 Productions

The organization of the “Productions” is still under discussion. Today’s idea is to have a “Computing Operations Group” responsible for the overall operations at Tier-0, Tier-1 and Tier-2.

The relative priorities of various productions and activities will be decided by a “Production Management Board”. A “Support Team” will be in charge of ensuring that the necessary infrastructure is in place and provides support to the users.

All centrally organized operations are expected to be performed using the ATLAS Production System ‘ProdSys’.

6.4.1 Distributed Production and Analysis Operations

While the operations at Tier-0 will be done in a centralized way, in a dedicated cluster using the T0 Management System, it is foreseen that the operations at Tier-1s, like the reprocessing, or at Tier-2s, like Monte Carlo simulation, will be done in a distributed way, on the Grid, using the ATLAS Distributed Data Management system (DDM).

As described in the Computing Model, the ProdSys will use the resources available at the Tier-1s and Tier-2s in the most efficient way possible. It is assumed that:

- The ATLAS DDM, through its own services, will provide the necessary information to run a given application, such as the location of the input data; this assumes accessibility to the necessary Grid catalogue. It will also provide the services to access the conditions databases holding the conditions data (e.g. geometry, calibrations, alignments, etc.).
- The Grid information systems will provide information on the availability of resources, such as CPU and data storage, on all sites accepting the ATLAS VO. This assumes that the Tier-1s and Tier-2s have published the relevant information and are able to communicate to the service whether they can accept or refuse to run an ATLAS job.

The ProdSys will use this information to optimize its own work. For example, it will be responsible for splitting a given task into several jobs if necessary and deciding where these jobs should be executed:

- Whether the jobs should be run at sites where the data resides, avoiding data replication?
- Whether the jobs should trigger a replication of data through the DDM services in order to run where CPU resources are available.
- What should be the priority in which the jobs are executed.

It should be noted that this decision may not necessarily be done at the top level of the system but can be delegated to one of its components like a Grid Resource Broker for example.

At the end of a job, it will also be the ProdSys' responsibility to store the produced data at the best place, which could be a requested one, and to ensure that, if requested, the data is stored in a safe way. It is also responsible for ensuring that the relevant information is stored in the bookkeeping and catalogue databases. ProdSys will maintain the complete provenance of all data produced and provide an interface to make this available to the collaboration.

6.4.2 Management of the ATLAS Virtual Organization

A Virtual Organization (VO) is a collection of people, resources, policies and agreements belonging to a real organization, such as the ATLAS Experiment. The Virtual Organization mechanism provides a way to give authorization to the user during task instantiation. User credentials are stored in the VO and organized in groups. Each site, when receiving a request from one user, is then able to decide whether or not (and with which priority) to give access to the underlying resources, on the basis of the information retrieved from the VO to which the user belongs.

Currently the VO is implemented as an LDAP database, hosted at NIKHEF. This database is synchronized with a VOMS server (Virtual Organization Management Service), which keeps track of the user roles and groups in a more efficient way. However, on account of some incompatibility between the European VOMS server and the US server, it is currently impossible to switch completely to the VOMS system.

In order to be able to access grid resources, members of the ATLAS Collaboration must also be part of the ATLAS Virtual Organization.

6.4.2.1 The Authorization System on the Grid

From the authorization point of view, a Grid is established by enforcing agreements between resource providers (facilities offering resources to other parties, according to a specific "common understanding") and VOs, where, in general, both parties control resource access.

The authorization mechanism is implemented taking into account two types of information:

- General policies, *i.e.* the relationship of the users with their VO: groups they belong to, roles they are allowed to play and capabilities they are allowed to exercise.
- Local policies of the resource provider: what users are allowed to do, ACLs, etc.

Authorization, as stated before, is based on policies written by VOs and their agreements with resource providers. It is the resource providers who enforce the local authorization policy. A VO can have a complex structure with groups and subgroups in order to clearly divide its users according to their tasks. Moreover, a user can be a member of any number of these groups.

A user, both at VO and group level, may be characterized by any number of roles and capabilities. The enforcement of these VO-managed policy attributes (group memberships, roles, capabilities) at the local level descends from the agreement between the VO and the resource provider. The latter can always override the permission granted by the VO, for example to ban unwanted users.

In ATLAS we foresee several different groups covering the various activities, software and computing, physics, combined performance, detector. Currently we have identified three different roles:

- Grid software administrator, in charge of installing and managing the resources.
- Production managers, responsible for official productions.
- Normal users.

6.4.2.2 ATLAS VO Structure and Responsibilities

A team of managers operates the ATLAS VO. Currently the people involved are:

- One general Manager and Coordinator;
- One manager for each of the three Grids: LCG/EGEE, Grid3/OSG and NorduGrid.

The VO managers perform controls on the eligibility of the users before admitting them to the ATLAS VO. The checks are generally performed by using the CERN HR database and/or with the help of known people who may act as guarantor for new users.

The VO managers usually inform the users about the status of the request, if it is either accepted or rejected, upon the completion of the checks. The VO administrator has the right to ban a user from the VO at any time if he fails to comply with the usage guidelines.

6.5 Experience with Data Challenges and other Mass Productions

To prepare for data taking in 2007, the LHC Computing Review [6-1] in 2001 recommended that the LHC experiments should carry out data challenges (DCs) of increasing size and complexity. A data challenge comprises, in essence, the simulation, done as realistically as possible, of data (events) from the detector, followed by the processing of those data using the software and computing infrastructure that will, with further development, be used for the real data when the LHC starts operating. The goals of the ATLAS Data Challenges are the validation of the ATLAS Computing Model, of the complete software suite, of the data model, and to ensure the correctness of the technical computing choices. In addition, the data produced allowed physicists to perform studies of the detector and of different physics channels.

The preparation of the simulated data consists in the generation of a large number ($>10^7$) of simulated "events". Each "event" is the result of a collision between two protons, and the full simulation requires the following steps:

- "Event generation": particles produced in the proton–proton collision are generated using a program based on physics theories and phenomenology. Several such programs are available, and are referred to as "event generators", or simply "generators".
- "Detector simulation and digitisation": the produced particles are transported through the detector elements according to the known physics laws governing the passage of particles through matter. The resulting interactions with the sensitive elements of the detector are converted into information similar to, and in the same format as, the digital output from the real detector in the "digitization" step.

Because of the very high intensity at which the LHC will operate, many proton–proton collisions will usually occur during the sensitive time of the particle detectors. Hence the digitized output from the ATLAS detector will contain the "piled-up" signals from the particles produced in several collisions. This leads to the next step:

- "Pile-up": the digitized signals from several simulated collisions are combined to produce events with realistic signal pile-up.

These events can now be used to test the software suite that will be used on real LHC data:

- "Reconstruction": the events are reconstructed to determine particle trajectories and energies from the detector data.
- Production of data for analysis: quite often this step is combined with the reconstruction one.

The data fed to the reconstruction step are, as far as possible, indistinguishable from real data, with one important exception: since the original information on the particle trajectories and energies is known from the event generation step, this information ("truth") is also recorded so that the output from the reconstruction step can be compared with it.

Up to now two Data Challenges have been run, DC1 in 2002 and 2003, and DC2 in the second half of 2004, followed in the first half of 2005 by a large-scale production to provide data for physics studies in view of the ATLAS Physics Workshop in June 2005.

6.5.1 Data Challenge 1

ATLAS Data Challenge 1 ran from spring 2002 to spring 2003. The main goals of DC1 can be summarized as follows:

- Put in place the full software chain;
- Deploy a large-scale production system;
- Provide data for the High-Level-Trigger community in view of their Technical Design Report;
- Build the ATLAS DC community.

Ten million physics events and 40 million single-particle events for a total volume of about 70 TB were produced using 21 MSI2k-days. Forty institutes in 19 countries actively participated in the effort. A full report on the exercise can be found in Ref. [6-6].

It appeared clearly during DC1 how important it is to have efficient procedures to deploy and certify the various components of the software, to validate the many sites which participated in the enterprise, and to monitor the quality of the data which was produced. The Distribution and Validation kits which are now part of the standard ATLAS infrastructure come directly from the DC1 experience.

During DC1 we have seen the emergence of production on the Grid. The production was done exclusively on the Grid in NorduGrid, it was partially done on the Grid on the US test-bed, especially in the phase 2 of DC1, and serious testing was done with prototypes of EDG (European DataGrid) middleware in close collaboration with the EDG developers. The lessons learned were extremely important for the development of the Grid projects.

ATLAS DC1 has proved to be a very fruitful and useful enterprise, with much valuable experience gained, providing feedback and triggering interactions between various groups, for example groups involved in ATLAS computing (e.g., HLT, offline-software developers, Physics Group), Grid middleware developers, and the CERN IT Department.

One of the most important benefits of DC1 has also been to establish a very good collaborative spirit between all members of the DC team and to increase the momentum of ATLAS computing as a whole.

6.5.2 Data Challenge 2

One of the main lessons learned from DC1 was that the production system needed to be more automated. The system was redesigned in summer 2003 and developed in the following months. The system is described in Chapter 5 of this document.

In parallel the Grid middleware was evolving and it became clear that ATLAS had to use different Grid flavours, as they were becoming more mature.

It was then decided that Data Challenge 2 (DC2) should concentrate on the following aspects:

- Deploy and use the new production system;
- Use in a coherent way three of the existing Grid flavours: LCG, Grid3 in US and NorduGrid in Scandinavian and associated countries;

- Perform a test of the operations that should be done in summer 2007 when the real data will be delivered by the detector. This test is called Tier-0 exercise.

The scale of the exercise was defined to produce at least the same amount of data as for DC1; that means fully simulate of the order of 10 million events.

The components of the software chain were the same as for DC1 with the following important differences:

- All software components were the newly developed ones based on object-oriented technology. They were developed either by the ATLAS collaboration or by other projects like the Geant4 simulation or the OO persistency system POOL, both developed within the LCG Applications Area project.
- In order to perform the Tier-0 exercise in conditions close those of the real situation in 2007, it was decided to add a new step in the production chain, called event mixing, in which different physics channels were mixed in such a way that the reconstruction program receives a realistic mixture of event types.
- As part of the Tier-0 exercise it was also foreseen to build the TAG collections after reconstruction of the events, and to distribute in real time both the ESD and AOD data to the Tier-1s as defined in the Computing Model.

6.5.2.1 Phase 1 (Event Generation and Detector Simulation)

Phase 1 was run only on the Grid using the three available flavours: LCG, Grid3 and NorduGrid. Four types of jobs were run: first the event generation followed by the Geant4 simulation, then the digitization and/or the pile-up (coupled with the digitization) of the data. The output data of each step were persistified in POOL format and registered in the Grid catalogues.

6.5.2.2 Phase 2 (Event Mixing)

Phase 2 consisted of two steps. In the first one the digitized data, produced worldwide, was concentrated at CERN. In the second step the events of different physics channels, which were produced independently, were mixed in order to have runs with a realistic mixture of physics events (as would be produced at the output of the Event Filter). The input of the process was the POOL digitized data; the output was either RDOs (Reconstruction Data Objects) in POOL format or ByteStream (BS) data equivalent to raw data. Typically ten input files were used producing one output file, events were picked up randomly in an *ad hoc* proportion. Pile-up was run in parallel on a subsample of more than two million events.

6.5.2.3 Phase 3 (Tier-0 Exercise)

Phase 3 was supposed to be a simulation of what will be the production at Tier-0 in 2007. In a first step each RDO or BS file should have been reconstructed producing Event Summary Data (ESD) in POOL format. In the second step ESD files should have been read and Analysis Object Data (AOD), condensed useful information for analysis by the physicists, should have been produced again in POOL format. Finally in a third step we wanted to stream the AODs in ten different physics channels and produce event collections to be used by the different physics groups.

6.5.2.4 Running Experience

- Phase 1: From July to mid September 2004 for the Geant4 simulation and in October and November 2004 for digitization, 10M physics events were produced, corresponding to 40 TB of data in 200 000 files. The three Grids were used roughly in the ratio LCG/Grid3/NorduGrid: 40/30/30% with a global efficiency of the order of 60%. In addition several million single-particle events and calibration events were produced in parallel. Since, as explained in the DC1 section, the pile-up step needed to use several input files, it was decided, for practical reasons, to run the process in a reduced number of sites where the "minimum-bias" and "cavern background" files were replicated in advance and to reduce the production to a subset of 3.5 million events.
- Phase 2: Started in mid November 2004. Data transfer to CERN was done using the ATLAS data management system DonQuijote. In the initial phase the transfer rate was of the order of 2–3000 files per day, that means 500 GB/day. After some improvements the transfer rate increased to 10 000 files per day (1.5 TB/day).
- Phase 3: For technical reasons the scope of the Tier-0 exercise was reduced, partly because some components of the software were not ready, partly because the resources at CERN were not fully available (due to the concurrent move to the new SLC3 operating system), and the need not to delay the start of the mass production for the physics workshop in June 2005 (Section 6.5.3). The exercise was reduced to a "proof of principle" and will be resumed in Summer 2005.

During the full period of DC2 several problems were encountered. During Phases 1 and 2 the following main problems were identified:

- Stagein/out, file transfer and file registration problems;
- Degradation of the response time of the central production database (Oracle);
- Problems with site (mis)configuration, or temporary faults, such as losing connection between worker nodes and the shared filing system where ATLAS software is installed;
- Problems with the LCG information system, leading to jobs going to the wrong queue;
- Connections lost with the LCG Resource Broker, leading to "orphan" files that could not be registered;
- Slowness of the LCG Resource Broker which limited the number of jobs we could run per day.

During Phase 3 we suffered also from problems with the POOL catalogue due to incompatibility of the LCG middleware and of the ATLAS software, as different versions were used and we had to set up the configuration by manual intervention. That showed the importance of a good synchronization in the development of software components used in different domains.

6.5.2.5 Conclusions

We started the DC2 production with a production system that was still under development and we discovered a lot of problems in the early days.

In addition, the development status of the services of the Grid caused troubles while the system was in operation. For example the Globus RLS (used by Grid3 and NorduGrid), the LCG Resource Broker and the information system were unstable during the initial phase.

Especially on the LCG Grid, we suffered from the lack of a uniform monitoring system and from the mis-configuration of sites and site-stability related problems.

Human errors, such as "expired proxy", and bad registration of files caused some additional troubles. We experienced network problems that caused the loss of connection between processes and Data Management System problems (e.g. loss of connection with the mass storage system).

Nevertheless we succeeded in running a large-scale production only on the Grid, using three Grid flavours, using an automatic production system that makes use of Grid infrastructure. Several "10 TB" of data were produced and moved among the different Grids using DonQuijote (ATLAS Data Management) servers. More than 200 000 jobs were submitted by the production system and we succeeded in running more than 2500 jobs per day.

6.5.3 Production for the ATLAS Physics Workshop

Immediately after DC2 we started the production of data for the ATLAS Physics community in view of the physics workshop in Rome in June 2005. That production was divided into one preparation phase (0) and 4 main phases:

- Phase 0: Event generation;
- Phase 1: Geant4 simulation, digitization and reconstruction;
- Phase 2: Replication of ESD, AOD and CBNT to CERN;
- Phase 3: Analysis;
- Phase 4: Pile-up.

Except for Phase 0 the production was run using the ATLAS production system ProdSys. The production model is very close to what was done for DC2.

Phase 0 was run by physicists on non-Grid systems. The files were registered in the Grid catalogue and replicated on the Grid(s) where they should be used later on. The manual registration into the Production System led to human errors, which had to be subsequently corrected, and should be avoided in future.

Phase 1 was run on Grid only using the three flavours: LCG, Grid3 and NorduGrid. Three types of jobs were run: first the Geant4 simulation, then the digitization of the data, and finally the reconstruction of the data producing ESD, AOD and CBNT (Combined Ntuple) in the same job. For each of the three steps the output data was persistified in POOL format and registered in its corresponding Grid catalogue, except the CBNT output that is in ROOT format. From end-January to end-March 2005 more than 5 million events were produced.

Phase 2, the concentration of the data at CERN was done in April and May 2005, using the ATLAS Data Management System DonQuijote. File transfer rates were limited by the performance of the Castor mass storage system at CERN.

Phase 3: The analysis scenario is still being developed. The baseline model is to use both AODs and TAG collections. In a first step the number of AOD files is reduced (in a ratio of 20:1) by concatenation. At the same time we produce TAG collections that will then be used for analysis.

Phase 4: It is intended to prepare some pile-up samples at a luminosity of 10^{33}. Three types of input files are needed: the signal, the minimum-bias, and the cavern background. These files have been produced and replicated to a limited number of sites that have enough resources to perform the exercise. It should be noted that the pile-up has been done at a luminosity ten times lower than the one used for DC2, in practice that means that the process is simpler and less stressing, in term of resources and I/O, than the same process run in DC2.

The four phases of the production were run in parallel, therefore we had at the same time in the system long jobs (simulation) and short jobs (reconstruction).

The ATLAS Production System (ProdSys) was run in the same way as for DC2 with a notable difference since we decided to use in addition a new version of the LCG executor using Condor-G to distribute jobs instead of the LCG Resource Broker. The net effect was an increase in the number of jobs running in parallel on LCG and a better use of the available resources. On the best day we ran more than 12 000 jobs.

More than 500 000 jobs were run, producing 6.1 million events fully simulated and reconstructed in 173 different physics samples, some of them with pile-up.

Despite the fact that we are trying to consolidate the production system, we have still many errors. The fraction of errors as measured for the LCG Grid is given in Table 6-1. It is computed as the ratio of the number of jobs failed for one specific error source over the total number of submitted jobs.

Table 6-1 Production for ATLAS Physics workshop failure rates on the LCG Grid.

System	Causes	%	Comments
Workload management		2.7	
Data management	stagein	25.5	~11.2% timeout ~7.3% connection to information system lost
	stageout	0.7	
ATLAS or LCG Grid configuration	Athena crash	4.3	Often “maxtime” not well defined
	User proxy expired	0.5	
	Site configuration	0.7	
Unclassified		5	

Owing to the very high number of files produced we experienced problems with the CERN Castor mass storage system. The catalogue was so large that it became inefficient. With the help of the CERN Castor team we succeeded in keeping the system running. We also had problems with the EDG/LCG Replica Location Service which were solved only after five days of intensive debugging.

All these problems show how important it is to stress-test in due time all the components of the system.

6.5.4 Combined Test Beam

Another important component of the evaluation of the Computing Model was the analysis of the Combined Test Beam (CTB) data.

A full slice of the barrel detector of the ATLAS experiment at LHC was tested for six months in 2004 with beams of pions, muons, electrons and photons in the energy range 1–350 GeV in the H8 area of the CERN SPS. The set-up included elements of all ATLAS sub-detectors:

- Inner Detector: Pixel, Semi-Conductor Tracker (SCT), Transition Radiation Tracker (TRT);
- Electromagnetic Liquid Argon Calorimeter;
- Hadronic Tile Calorimeter;
- Muon System: Monitored Drift Tubes (MDT), Cathode Strip Chambers (CSC), Resistive Plate Chambers (RPC), Thin Gap Chambers (TGC).

Ninety million events, with an average size of 50 kB/event, were collected and stored on Castor, for a total of about 4.5 TB.

It has been a challenging pre-commissioning exercise: for the first time, the complete software suite developed for the full ATLAS experiment was used for real detector data. Important integration issues like combined simulation and reconstruction, connection with the online services and management of many different types of conditions data were addressed for the first time.

In order to manage simulation, digitization and reconstruction of such a large data volume, we adopted, for large-scale CTB operations, the production systems that had been developed for the Data Challenges:

- for the applications that need access to the conditions database and are performed at the CERN LSF batch system (reconstruction, both of real and simulated data, and digitization of simulated data) we used the DC1 production infrastructure;
- for the bulk production of simulated data on the Grid we used the production infrastructure developed for DC2 and for the Rome production.

6.5.4.1 Production Activities for Real Data

For book-keeping of the reconstruction of real data we used the AMI metadata catalogue and the AtCom [6-7] job submission system, with some new features to cope with CTB requirements.

Large-scale reconstruction for real data has already been performed twice in 2005 with two different software releases. In both cases a large sample of good runs (about 400, for a total of about 25 million events) was processed, half of them with the fully combined set-up and half of them with combined calorimetry only. See the CTB Offline Web Page [6-8] for more details.

The average CPU time per event is of the order of 1.5 kSI2k-seconds. All the runs were split into logical files of maximum 10 000 events each and for each job three types of output files were produced and stored on Castor:

- ROOT files with the combined n-tuples (20 kB/event);
- POOL files with ESDs (40 kB/event);

- XML files storing the POOL file catalogue needed by the ESD files.

These productions were rather successful, with an overall failure rate of about 2%. New reprocessing of these data is already foreseen later in 2005 with the next major ATLAS software release.

6.5.4.2 Production Activities for Simulated Data

The Monte Carlo production for the 2004 Combined Test Beam started in September 2004. The production was divided in two parts: a preproduction phase and a production phase. The preproduction phase started in September 2004 and lasted until April 2005. During this period, Monte Carlo events were simulated, digitized and reconstructed using AtCom on the CERN LSF batch system and the AMI database was used for book-keeping. The goal of this production was twofold:

- Test the different software packages that were part of the simulation, digitization and reconstruction in order to obtain a stable version;
- Compare the first Monte Carlo reconstructed events with those produced by the reconstruction of real data.

The events were processed using different ATLAS software releases from 8.6.0 to 10.0.2. More than 500k electron, 900k pion, 350k muon and 10k proton events, for a total of >1.7 million events, were generated. The statistics of the simulated, digitized, and reconstructed samples is slightly different, mostly due to the fact that the software was being developed and not all detector components were available in a given release. The mean processing time per event ranged between 1.7 kSI2k-seconds for electron simulation and 1.0 kSI2k-seconds for pion reconstruction.

In May 2005 we started the production phase for a sample of 392 runs, corresponding to almost 4 million events, with the same conditions of the good "real" runs selected by the CTB groups. For that reason, we adopted the tools used for Rome production in order to cope with such a big production. For the simulation, job definition entries were filled in the ATLAS development database. We used the standard LCG Grid productions tools (the Windmill supervisor and the Lexor executor). ATLAS release 10.0.1 was used for the simulation together with a few additional software package tags. Once the jobs finished successfully, the output files were replicated to the CERN Storage Element.

The digitization and reconstruction of those events is being done at CERN using AtCom and ATLAS release 10.0.2. This part of the processing is not being submitted to the Grid since the CTB Conditions Database is not replicated worldwide.

More information about the produced samples can be found in the CTB Simulation web page listed in Ref. [6-9].

6.5.4.3 Conclusions

The main problem that we had during these large-scale productions is that, even for stable and "validated" releases, a lot of "harmless errors" were issued during reconstruction and we had to wait for final answers from many developers before starting each production. This is due to the fact that usually the developers do not test the code on large statistics. So the "final" validation of reconstruction in a given release is quite painful.

The experience with AMI and AtCom has been quite positive: AMI is a useful book-keeping database and helps the end user to search for relevant information concerning the productions. It is of great help also for checking the production itself while it is runing. AtCom is a very practical tool for job submission but it cannot cope with the monitoring of a large number of jobs: job submission and monitoring should be done in groups of no more than 500 jobs at the same time. It was decided to use the Rome production Grid tools for CTB simulation on account of the large amount of runs that have to be simulated.

Concerning future activities, we know already that the simulation of data for the CTB will continue at least until the end of 2005 and that a full reprocessing of good real data will be done at least twice before the end of 2005, with two new major ATLAS releases, namely a new set of conditions data and improved combined reconstruction algorithms.

6.6 Summary

Based on the experience acquired in running the data challenges and other productions, ATLAS is now preparing for the real data-taking scheduled for mid 2007.

We plan to put in place a Computing Operations Group which will be in charge of the organization and the running of the operations at Tier0, Tier-1s and Tier-2s.

Within the ATLAS VO, several working groups will be created. The definitions of the roles and responsibilities are currently being discussed.

An improved Production System is being prepared and will be tested in the context of the Service Challenge 3 (SC3) in which ATLAS intends to run a Tier-0 exercise in autumn 2005. That exercise will allow the production at Tier-0 and the data transfer from Tier-0 to Tier-1s to be tested. By the end of 2005, we foresee running a distributed Monte Carlo production, which will allow us to test data transfers between Tier-1s and Tier-2s, in both directions. The next step will be to run Data Challenge 3 in early 2006; that operation will be considered as a Computing System Commissioning.

Finally, Monitoring and Accounting are two important pieces for running the ATLAS productions in an efficient way. We intend to unify the existing systems assuming that the LCG Project could help us in this area.

6.7 References

6-1 S. Bethke *et al.*, *Report of the Steering Group of the LHC Computing Review*, CERN-LHCC-2001-004, March 2001

6-2 ATLAS software installation, https://uimon.cern.ch/twiki/bin/view/Atlas/InstallingAtlasSoftware

6-3 Pacman, http://physics.bu.edu/pacman/

6-4 https://pc-ads.roma1.infn.it/atlas_install/index.php

6-5 NorduGrid web site, http://www.nordugrid.org

6-6 R. Sturrock *et al.*, *A Step Towards A Computing Grid For The LHC Experiments: ATLAS Data Challenge 1*; CERN-PH-EP-2004-028

6-7 AtCom website, http://atlas-project-atcom.web.cern.ch/atlas-project-atcom/

6-8 ATLAS test beam website, http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/testbeam/testbeam.html

6-9 ATLAS test beam simulation, http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/testbeam/simulationCTB/productionMC.html

7 Resource Requirements

7.1 Introduction

The primary purpose of the Computing Model exercise is to describe the model and estimate the computing resources required for a full year of data taking in 2008, with the inputs as described in Section 2.2. Two main areas of computing activity have to be considered:

- Large-scale production:

  E.g. reconstruction, MC simulation. Two processing passes are assumed for the dataset in a year; the model is as described in Section 2.2.5.5

- Data analysis.

  The data analysis is more uncertain in its requirements, but a plausible scenario has been outlined in section Section 2.3. It should be stressed that the model and the resource requirements below do not include the personal analysis performed at the Tier-3 resources (or their equivalent share of Tier-2s and Tier-1s). The user load includes a CPU-intensive term that is in aggregate equal to the simulation of 20% of the data rate, (this may be full simulation, fast simulation, reconstruction or other CPU-intensive work), another term that describes the analysis of the working group DPDs in a chaotic fashion, and a final contribution for limited event re-reconstruction. Given the many different analyses to be performed and the various data formats required, these are by necessity approximations.

It is important to note that the predictions that follow include allowance for inefficiencies in the usage of CPU and disk resources. For scheduled CPU usage, the efficiency is assumed to be 85%, while for chaotic usage it falls to 60%. The disk storage efficiency is taken to be 70%, while tape storage efficiencies are assumed to be 100%.

The model assumes there will be 10 Tier-1s, which will be of different sizes. We assume there are on average three Tier-2 facilities for each Tier-1; again, there will be a range of sizes of Tier-2. The CERN Analysis Facility is taken to have 100 active ATLAS users (in addition to the active 600 users associated with the Tier-2s), five times the size of a nominal Tier-2.

The simulation is (along with the analysis load) a key driver of the Tier-2 resource requirements. It also has an effect on the Tier-1 storage and CPU requirements, as they host the simulated data. Many previous experiments have been severely limited in their capacity to produce fully simulated data, and this has hampered their physics output. Table 7-1 shows, for the year 2008 data only, the components that depend on the percentage of the data that is fully simulated as separate items for the case of 20% of the full data rate simulated; this simulation rate (4×10^8 events simulated per year) would be barely acceptable. Table 7-2 gives the same breakdown for the full capacity required at the start of 2008, i.e. including the capacity to deal with the 2007 data as well.

It has been assumed that the Tier-1 facilities follow the processing model outlined in the Offline Computing Model. It is further assumed that the raw data are stored on tape/slow access media, with only a small subset remaining on disk to allow software development and testing[1]. The actual Tier-1 and Tier-2 capacity might be larger, and indeed shared with other experiments. Here only that part that is visible to and accessible by all ATLAS members, is taken into account, and would be credited in the ATLAS accounting.

Table 7-1 The projected resources for handling the 2008 data alone, assuming full simulation of 20% of the p-p data rate.

	CPU (MSI2k)	Tape (PB)	Disk (PB)
CERN Tier-0			
Simulation	0	0.0	0.0
Other	4.1	4.7	0.4
CERN AF			
Simulation	0	0.	0.0
Other	2.3	0.5	1.8
Tier-1s			
Simulation	2.8	1.3	1.7
Other	16.7	5.9	11.7
Tier-2s			
Simulation	5.9	0.0	1.0
Other	11.6	0.0	6.1

Table 7-2 The projected total resources required at the start of 2008 for the case when 20% of the data rate is fully simulated.

	CPU (MSI2k)	Tape (PB)	Disk (PB)
Tier-0	4.1	5.7	0.39
CERN AF	2.7	0.5	1.9
Sum of Tier-1s	24.0	9.0	14.4
Sum of Tier-2s	19.9	0.0	8.7
Total	50.6	16.9	25.4

7.2 Ramp-up and Resource Requirement Evolution

Clearly, the system described by Table 7-1 will not be constructed in its entirety by the start of data taking in 2007/2008. From a cost point of view, the best way to purchase the required resources would be 'just in time'. However, the early period of data taking will doubtless require much more reprocessing and less compact data representations than in the mature steady state. There is therefore a requirement for early installation of both CPU and disk capacity.

It is therefore proposed that by the end of 2006 a capacity sufficient to handle the data from first running under the conditions described in Section 2.2, needs to be in place, with a similar ramp-

1. This defines one extreme position, assuming that almost no reprocessing will be possible at CERN. In reality, CERN should be able to provide significant resources for reprocessing during non-running periods.

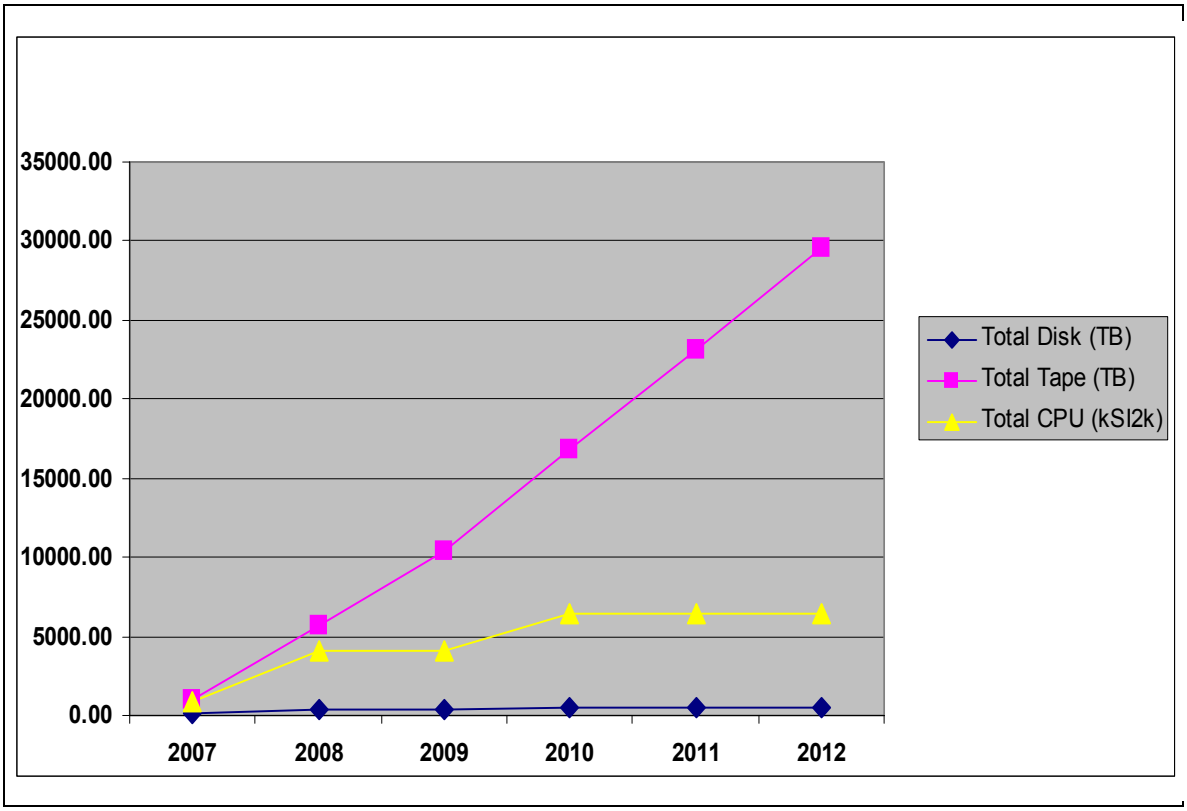

Figure 7-1 The projected growth in ATLAS Tier-0 resources with time.

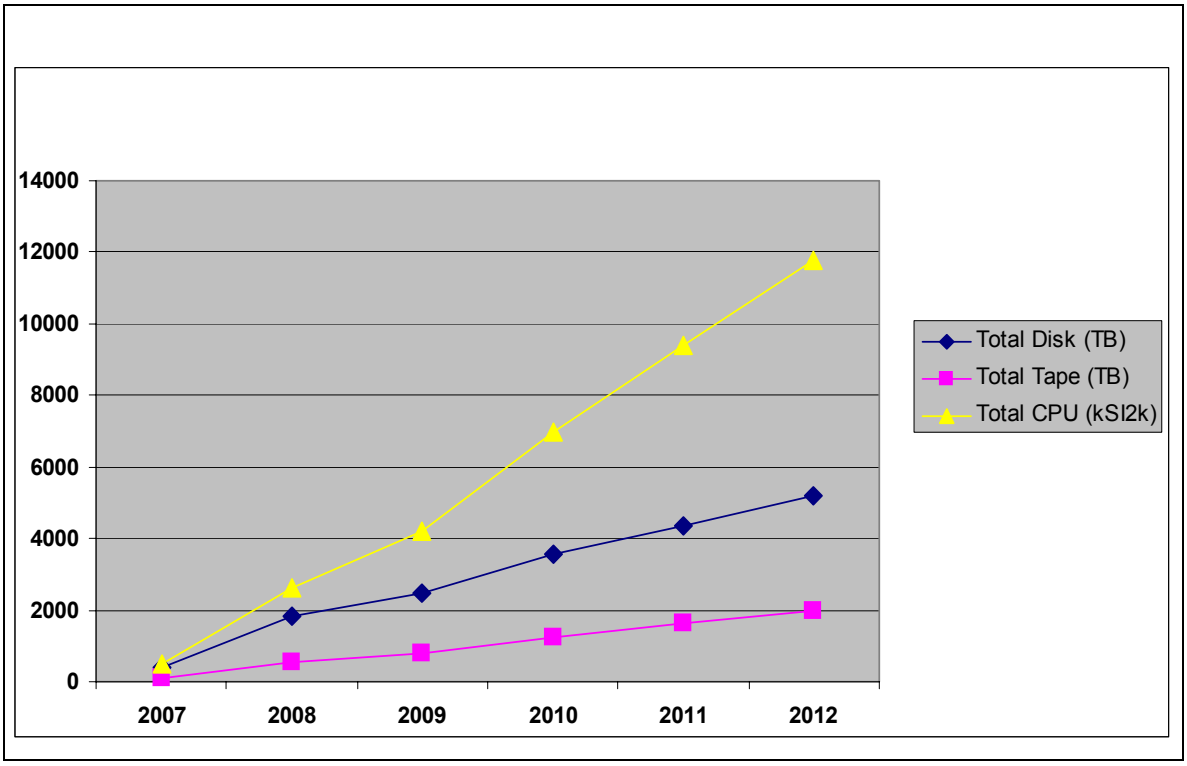

Figure 7-2 The projected growth in the ATLAS CERN Analysis Facility.

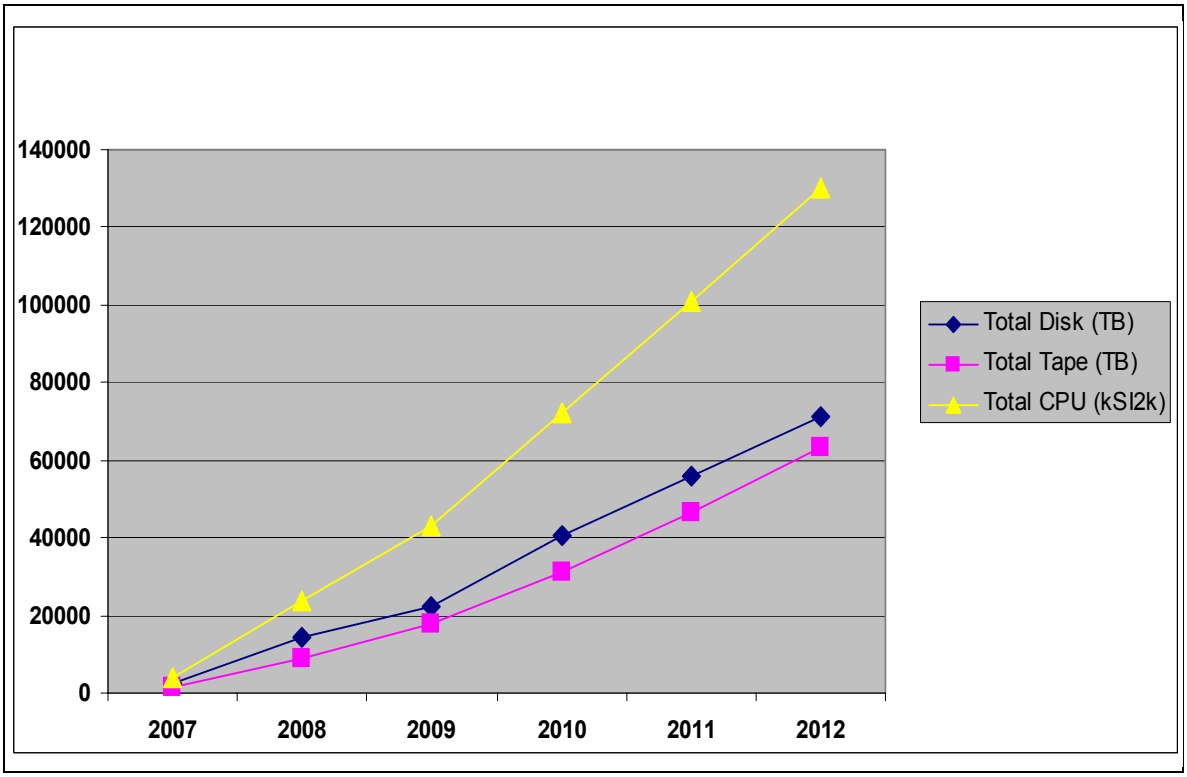

Figure 7-3 The projected growth in the capacity of the combined ATLAS Tier-1 facilities

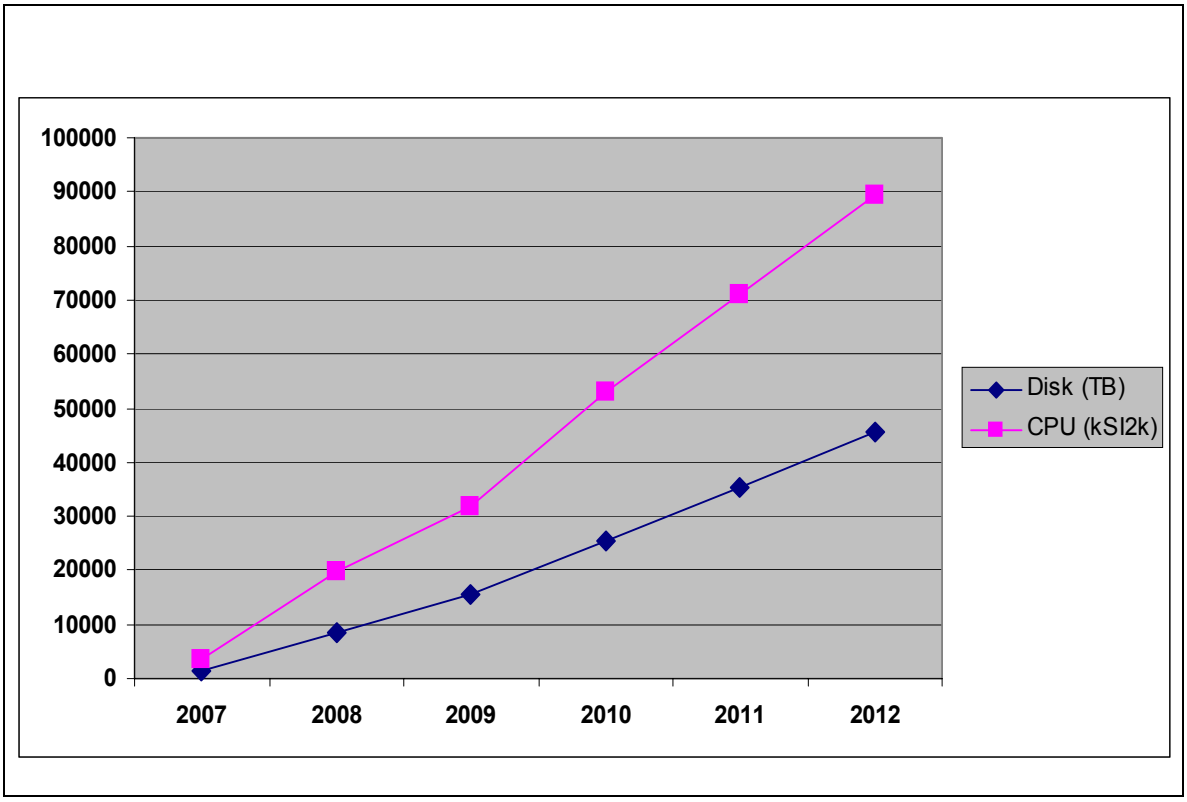

Figure 7-4 The projected growth of the combined ATLAS Tier-2 facilities.

ing up of the Tier-1 facilities. During 2007, an additional capacity required for 2008 should be bought. In 2008 an additional full-year of capacity should be bought, including the additional archive storage medium/tape required to cope with the growing dataset. This would lead to a capacity, installed by the start of 2008, capable of storing the 2007 and 2008 data as shown in Table 7-2; the table assumes that only 20% of the data rate is fully simulated.

For the Tier-2s, a slightly later growth in capacity, following the integrated luminosity, is conceivable provided that the resource-hungry learning-phase is mainly consuming resources in Tiers 0 and 1. However, algorithmic improvements and calibration activity will require also considerable resources early in the project. As a consequence, we have assumed the same ramp-up for the Tier-2s as for the higher Tiers.

Once the initial system is built, there will for several years be a linear growth in the CPU required for processing, as the initial datasets will require reprocessing as algorithms and calibration techniques improve. In later years, subsets of useful data may be identified to be retained/reprocessed, and some data may be rendered obsolete. However, for the near future, the assumption of linear growth is reasonable. For storage, the situation is more complex. The requirement exceeds a linear growth if old processing versions are not to be overwritten. On the other hand, as the experiment matures, increases in compression and selectivity over the stored data may reduce the storage requirements.

The project growth in the required resources in the various Tiers are shown in Fig. 7-1–Fig. 7-4. The projections do not include the replacement of resources, as this depends crucially on the history of the sites at the start of the project.

7.3 Networking Requirements

The required bandwidth out of CERN is essentially set by the migration of data to the external Tier-1 facilities. There is a need for semi-immediate (real time) transfer of data offsite.

It should be noted that the future cost evolution for bandwidth depends on many external factors. Thus far, we have benefited from uncharged access to bandwidth provided under external funding. It is hoped that this continues, but as we are dependent on bandwidth, we are carrying a large potential source of financial risk.

The traffic is based on the planned flow of RAW, ESD and AOD data from CERN to the Tier-1; this is assumed to be direct point-to-point. As there are expected to be distinct data-taking periods for p-p and heavy ions running, the nominal aggregate bandwidth is given in the two cases in Table 7-3 and Table 7-4 respectively.

A projection of the likely time profile of the nominal traffic between Tiers is given by Figure 7-5-Figure 7-7. All of the quoted bandwidths correspond to the rate aggregated over the month, with no efficiency or other factors applied. At the least, some additional capacity will be needed for the peak load and 'catch-up' situations.

A very high networking requirement between CERN and the Tier-1 cloud is indicated. There is also a high bandwidth required between each Tier-1 and the rest of the Tier-1 cloud. The precise bandwidth required depends on the network topology assumed to link the Tier-1s; some routing options may also imply substantial fast disk buffers in the system.

Table 7-3 The decomposition of the traffic for pp data between CERN and an average-sized Tier-1 in 2008-2009.

Source	Inbound from CERN (MB/s)	Outbound to CERN (MB/s)
RAW	30.4	
ESD Versions	10	1.41
AOD versions	18	0.28
TAG Versions	0.18	0
Group DPD		0.81
Total CERN/Average Tier-1	58.58	2.51

Table 7-4 The decomposition of the aggregate nominal bandwidth for heavy ions data between CERN and an average heavy-ions Tier-1 in 2008-2009.

Source	Inbound from CERN (MB/s)	Outbound to CERN (MB/s)
RAW	128.8	
ESD Versions	7.5	1.72
AOD versions	5	0.33
TAG Versions	0.005	0.02
Group DPD		0.8
Total CERN/Average Tier-1	141.3	4.49

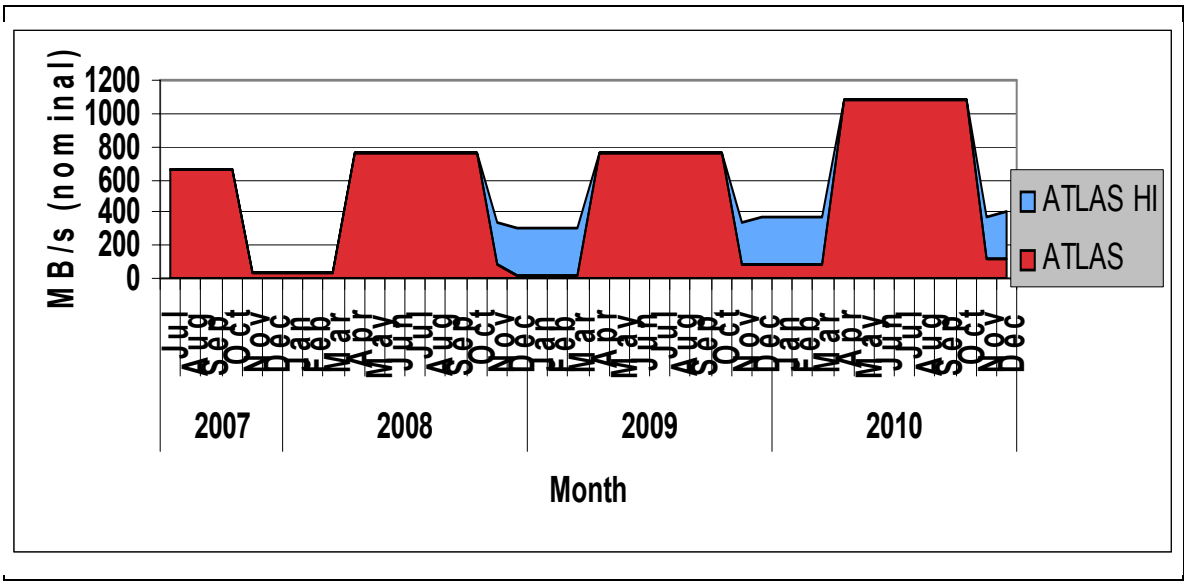

Figure 7-5 The projected time profile of the nominal bandwidth required between CERN and the Tier-1 cloud.

An estimate has also been made of the ATLAS-only traffic between the Tier-1 facilities; this is comprised of the copying of 'back-up' samples from another Tier-1 for the various data processing versions. It also includes the transfer of analysis-group derived datasets. The actual traffic from other sources, such as the data flow associated with job submission is not included, but believed to be relatively small. Naturally, factors for efficiency of bandwidth usage and contention need to be added. The aggregate nominal bandwidth required for 2008 is given in Table 7-5, while the projected time profile is given in Figure 7-6.

Table 7-5 The decomposition of the Tier-1 to Tier-1 traffic for an average sized Tier-1 in 2008-2009.

Source	Inbound from other Tier-1s (MB/s)	Outbound to other Tier-1s (MB/s)
ESD Versions	62.8	62.8
AOD versions	31.7	31.7
TAG Versions	0.3	0.3
Group DPD	36.7	36.7
Total Tier-1 to Tier-1	131.6	131.6

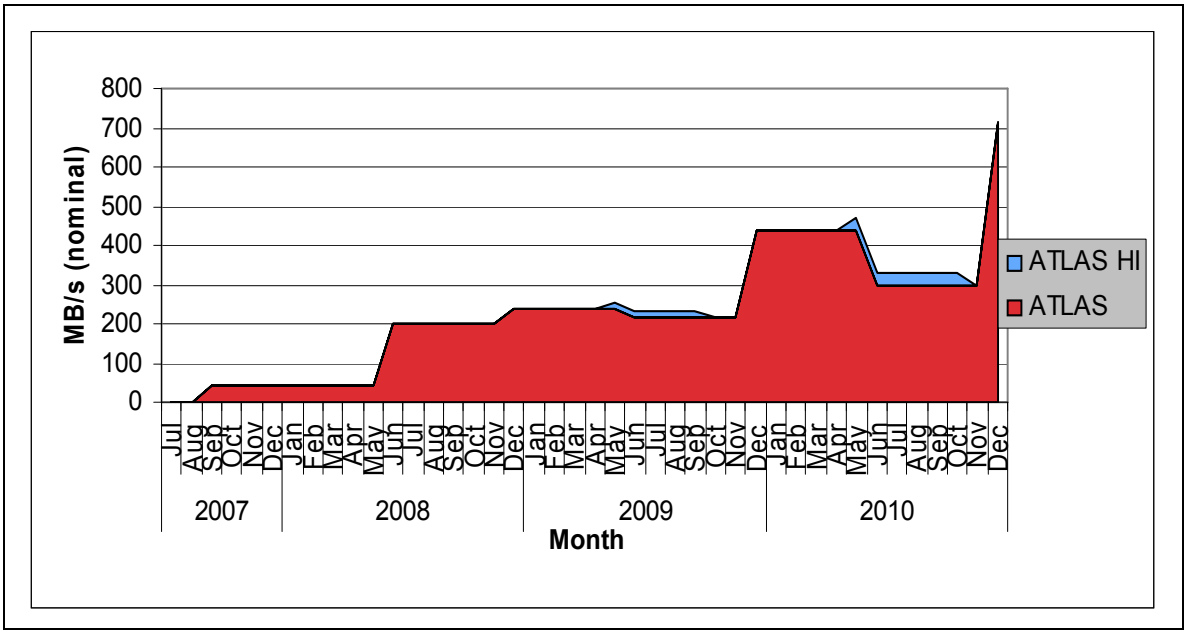

Figure 7-6 The projected time profile of the nominal aggregate bandwidth expected for an average ATLAS Tier-1 with other Tier-1s in the cloud.

The bandwidth required between a Tier-1 and Tier-2s will depend very much on the size of the Tier-2s concerned, but in terms of the files to be made available for general use at each Tier-2 it will be typically below 10 MB/s. The traffic associated with user jobs is again not included, and may well be the dominant term in this case. However, estimates of the expected traffic between an average Tier-1 and its associated cloud of three Tier-2s have been made; the projected time profile of the nominal aggregate traffic is given in Figure 7-7.

Remote Event Filter processing is not considered in this baseline model, but if it is to occur then a significant additional bandwidth upwards from 10 Gb/s will be needed between the ATLAS pit and the remote site, the actual capacity required depending on the Event Filter load displaced.

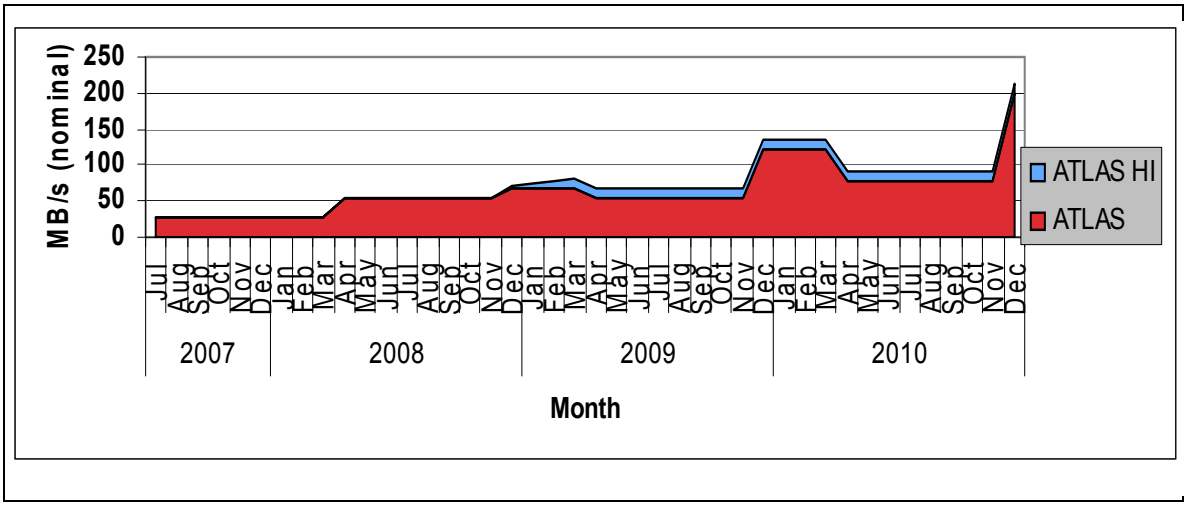

Figure 7-7 The projected time profile of the nominal aggregate bandwidth expected for an average ATLAS Tier-1 and its three associated Tier-2s.

8 Project Organization and Planning

8.1 Organizational Structure

The ATLAS Computing Organization was completely revised at the beginning of 2003 in order to adapt it to the current needs of the Software & Computing Project. The basic principles of the organization are the following:

- A Management Team, consisting of the Computing Coordinator and the Software Project Leader, coordinates all activities. The Software Project Leader has direct responsibility for software development, the Computing Coordinator for other computing activities. Both are members of the ATLAS Executive Board, of the Computing Oversight Board and of the International Computing Board.
- A number of relatively small bodies, with a clearly-defined mandate, act as executive or consultative committees.
- All bodies hold short (but frequent) meetings, in order to ensure a correct information flow both vertically and horizontally across the project structure.
- Interactions occur at all levels between the ATLAS Software & Computing Project and the LCG Project.

The bulk of the new structure (the Computing Management Board and the Software Project Management Board) was put in place during the first half of 2003; it was completed in autumn 2003 with the Operations, Data Challenges and Grid Management Board and in mid-2004 with the organization of the ATLAS-wide Database Project. The organization of the "Operations, Data Challenges and Grid" activity area was further revised in spring 2005. Figure 8-1 gives a pictorial view of the relations between the computing management bodies and major areas of activity.

After a collaboration-wide nomination procedure, the Computing Coordinator and the Software Project Leader are appointed by the ATLAS Spokesperson and confirmed by the ATLAS Collaboration Board for a two-year term. Re-appointments are possible but they require a two-thirds majority in the Collaboration Board.

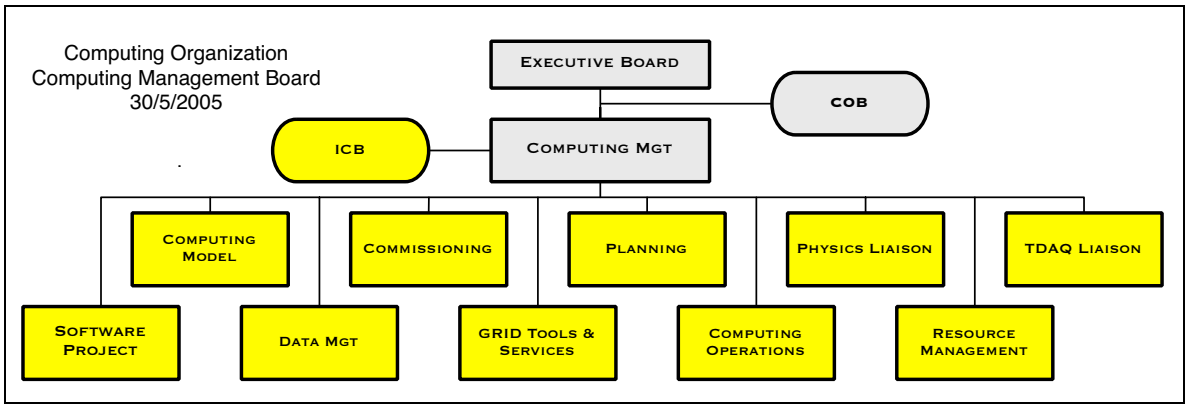

Figure 8-1 Composition of the Computing Management Board and relations with other computing management bodies (ICB and COB).

As it is important to keep a high level of communication at all levels with people involved in other ATLAS projects, their representatives are integrated in the Software & Computing Project organization and have dual reporting lines.

ATLAS Software & Computing Workshops take place four times per year (outside CERN approximately every 18 months). Workshops consist of plenary sessions at the beginning and the end of the week, and a number of parallel sessions in-between, where all activities and their integration issues are discussed. In addition, there are regular, more frequent meetings of all working groups.

8.1.1 The Computing Management Board

The Computing Management Board (CMB) is the overall project management body. It is chaired by the Computing Coordinator. The CMB meets every two weeks to discuss all matters that involve cross-coordination between different activity areas, global planning and scheduling, high-level milestones, external constraints, relations with other ATLAS projects and with the LCG project.

The CMB consists of the coordinators of all Software & Computing activity areas and liaisons from (and to) other, related, ATLAS projects:

- the Computing Coordinator (chair)
- the Software Project Leader
- the Database and Data Management Project Leader(s)
- the Grid Tools & Services Coordinator
- the Computing Operations Coordinator
- the Computing Resources Coordinator
- the Planning Officer
- the Offline Commissioning Coordinator(s)
- the International Computing Board Chair
- the Computing Model Working Group Chair
- the Physics Coordinator
- the TDAQ Liaison
- *ex-officio*, ATLAS Management (spokesperson and deputy spokesperson responsible for computing)

At the end of every Software & Computing workshop, four times a year, a joint meeting of the CMB and the SPMB (see Section 8.1.4) addresses global planning and scheduling issues.

8.1.2 The Computing Oversight Board

The Computing Oversight Board (COB) is a subcommittee of the ATLAS Executive Board that oversees the coherence of computing activities across the boundaries of the TDAQ Project, Computing Coordination and Physics Coordination. The COB is chaired by the Deputy Spokes-

person with responsibility for computing and comprises the Spokesperson and Deputies, the Computing Coordinator, the Software Project Leader, the TDAQ Project Leaders and the Physics Coordinator.

8.1.3 The International Computing Board

The International Computing Board (ICB) is the body that discusses the computing resources available to ATLAS, both hardware and manpower, and contributions from various collaborators to the ATLAS Computing project. As such the ICB receives and considers responses to effort and other computing reviews by the Computing Resource Review Board, CERN management and other such bodies.

It is the ATLAS forum for issues concerning the relationship between computing at the institutes and computing at CERN, and the means by which physicists in the institutes can contribute to the ATLAS computing project and analyse ATLAS data. The board considers and approves Software Agreements between Institutes and Funding Agencies and ATLAS, and also considers all disputes concerning their implementation.

It particularly concerns itself with issues regarding the construction and deployment of the worldwide computing infrastructure for ATLAS, and in particular relations between ATLAS and the LHC Computing Grid (LCG) project, and also network-based collaborative tools. As such, it appoints the ATLAS member of the LCG SC2 committee, and receives quarterly reports on the LCG project.

The ICB's remit also extends to those issues in the ATLAS software project that concern the outside institutes such as ease of installation of code, compilers, platforms, ease of use of the ATLAS suite at remote sites, licensing issues etc.

Given the importance of resources in its remit, the board is composed of delegated representatives of the funding agencies, as communicated by the National Contact Physicists, and a chairperson elected for a renewable two-year term by the voting members. The ICB has various *ex-officio* members to aid it in its work, namely the Computing Coordinator, Software Project Leader, Grid Coordinator(s), the ATLAS Spokesperson, Deputy Spokesperson and the ATLAS Resources Coordinator. The ATLAS-IT liaison and SC2 member appointed by the ICB are required to be in attendance.

The ICB meets four times per year, during Software and Computing Workshops.

8.1.4 The Software Project

The Software Project is led by the Software Project Leader. The mandate of the Software Project consists in the

- provision of the software framework, infrastructure and core services that are needed to simulate, reconstruct and analyse ATLAS data;
- coordination of the algorithmic software development activities that take place within detector and combined performance groups.

The Software Project Management Board (SPMB) meets every two weeks to discuss coordination and software development planning issues. Its membership consists of:

- the Software Project Leader (chair)
- the Computing Coordinator (*ex officio*)
- the Core Services Coordinator
- the Software Infrastructure Team Chair
- the Simulation Coordinator
- the Calibration & Alignment Coordinator
- the Event Selection, Reconstruction & Analysis Tools Coordinator
- the four detector Software Coordinators
- the Database and Data Management liaison
- the HLT Liaison
- the Physics Validation Coordinator
- the Combined Test Beam Offline Coordinator (in 2004–2005)
- in attendance, all members of the CMB
- *ex-officio*, ATLAS Management (spokesperson and deputy spokesperson responsible for computing)

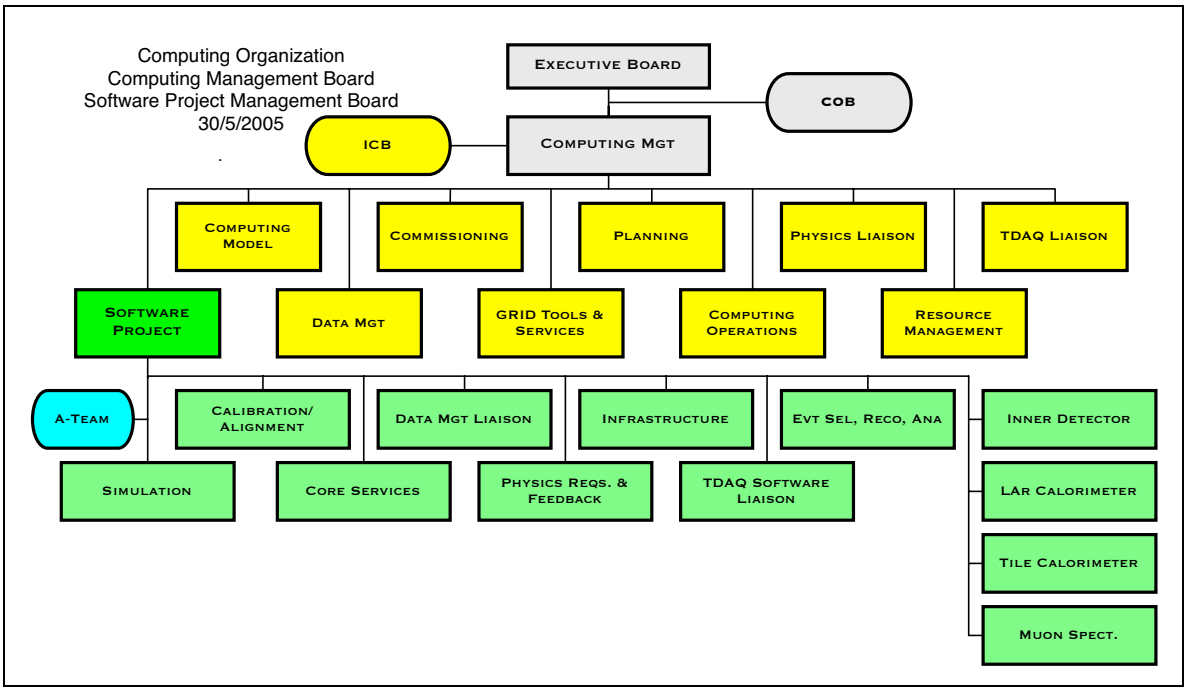

Figure 8-2 Composition of the Software Project Management Board (in the lower part of the figure) and its relations with other computing management bodies.

In addition to the SPMB, the Architecture Team, a consultative committee of core software development experts, advises the Software Project Leader on technical issues. Details of all software development activities are given in Chapter 3 of this document. Figure 8-2 gives a pictorial view of the areas of activity covered by the Software Project and its relations with the CMB and other computing management bodies.

8.1.5 The Database and Data Management Project

The ATLAS Database and Data Management (DB/DM) Project was set up in spring 2004 to lead and coordinate all database activities within the ATLAS Collaboration, including those previously under the responsibility of the Software & Computing Coordination, Technical Coordination, the TDAQ Project and the Detector Projects.

The mandate of the DB/DM Project includes the development and deployment of the software necessary to operate the databases holding the following types of information:

- detector geometry, construction and installation data
- survey data
- online configuration, run book-keeping and run conditions (from the Detector Control System and other sources of online data)
- event data and metadata
- online and offline calibrations and alignments
- offline processing configuration and book-keeping
- Grid access to event and non-event data (catalogues, replicas etc.).

Details of DB/DM development activities are given in Chapter 4 of this document.

8.1.6 Grid Tools and Services

This project groups all activities related to the definition of the Grid services that we need for the operation of our Computing Model, as well as the development and deployment of our production system and related tools. It includes the following activities:

- Development and deployment of the Production System for Tier-0 and world-wide operations (supervisor, executors, production database set-up and related tools, the connection to the book-keeping database and the connection to Distributed Data Management)
- Job preparation and submission, including provisions for production jobs and distributed analysis jobs on the Grid, as well as single test jobs
- Grid architecture and tools:
    - middleware contacts (liaison with middleware projects, transmission of requirements from us and information on new tools or functionalities from middleware projects to us)
    - testing of (new) middleware components
    - definition of the Grid tools and environment to be used by the production and analysis system and by interactive users
- Development and deployment of the Distributed Analysis System (provision of a system that allows the analysis of ATLAS data on the Grid, including job split and merge, submission and retrieval, book-keeping).

Details of Grid tools and services development activities are given in Chapter 5 of this document.

8.1.7 Computing Operations

This project groups all activities related to the efficient operation of the ATLAS computing system. It includes in particular the following activities:

- CERN Tier-0 operations from the output of the Event Filter to data distribution to Tier-1s, including calibration/alignment and reconstruction procedures
- World-wide operations:
  - simulation job distribution
  - re-processing of real and simulated data at Tier-1s
  - data distribution and placement
- Software distribution and installation
- Site and software installation validation and monitoring
- Coordination of Service Challenges in 2005–2006
- User Support for Grid computing issues.

Details of the activities related to computing operations, as well as of Data Challenges, are given in Chapter 6 of this document.

8.1.8 Computing Resources Management

This project covers all administrative and managerial aspects relative to ATLAS computing operations. In particular, it consists of the following work packages:

- Management of CERN resources (afs and disk space, castor, user accounts)
- CERN and world-wide resource planning and reservation for centrally managed computing operations
- Job accounting (collection of information on Grid jobs, development of book-keeping and tools to access accounting information)
- Job and Grid monitoring (development of tools for real-time monitoring of job and Grid performance)
- Global accounting of used CPU and data storage resources
- Security and Virtual Organization management (groups, user registration, quotas etc.).

8.1.9 Relations with other ATLAS-wide Projects

8.1.9.1 Relations with the TDAQ and HLT Projects

The software environment of the TDAQ High-Level Trigger and the offline system are tightly coupled, as they share the framework, the data store in memory and the HLT algorithmic code. The performance requirements of the HLT are clearly stronger than those of the offline users, therefore a close cooperation in the design of the common components is necessary. For this rea-

son, an HLT liaison person is present in the SPMB; the Architecture Team also includes TDAQ and HLT people.

Conditions and configuration data have to be exchanged between the online and offline systems. In order to ensure coherent developments, the DB/DM project oversees online applications of databases and includes in its steering group the persons responsible for their development.

The Computing Model Working Group includes a TDAQ liaison person whose task is to make sure that there is a consistent model, and implementation, for the flow of event and non-event data from the online to the offline system.

At a higher level, a TDAQ liaison person is a member of the CMB, and the Computing Coordinator, Software Project Leader and DB/DM Project Leaders are members of the TDAQ Steering Group. As outlined above, the COB also acts as a communication channel between the projects and with ATLAS management.

8.1.9.2 Relations with Physics Coordination

The aim of all software and computing development and operation activities is to enable all ATLAS physicists to analyse the data. It is therefore extremely important that there be a continuous flow of communication between the physics and computing communities, in one direction to provide requirements and feedback, and in the other direction to provide tools, information on what can be done and how, and user support.

At the highest level, the Computing Coordinator is a member of Physics Coordination, and likewise the Physics Coordinator is a member of the CMB; both are members of the COB. The Physics Software Validation Coordinator, appointed by the Physics Coordinator, is a member of the SPMB.

The Combined Performance Working Groups (e-gamma, jets, tau/etmiss, muons, flavour tagging) meet during Software & Computing Workshops in order to increase the contacts between people interested in physics performance and software developers. The ESRAT (Event Selection, Reconstruction & Analysis Tools) group (see Section 3.9) provides a discussion forum on detector and combined reconstruction algorithms and performance that involves HLT and offline developers as well as physics analysis users.

In addition, software and computing issues, as seen by end-users, are always treated among the topics of the ATLAS Physics Workshops.

8.1.9.3 Relations with Commissioning Activities

A Commissioning Coordination structure was set up in ATLAS at the beginning of 2004. It includes two Offline Commissioning Coordinators, who are also members of the CMB. Their task is to collect the requirements for offline software support by the detector groups, plan the necessary development activities and oversee their implementation. Liaisons towards the Commissioning organization have been appointed by all offline software detector groups. More details can be found in Section 3.12.

8.1.10 Relations with the LCG Project

There are interactions at all levels between ATLAS and the LCG project. The spokesperson is a member of the LCG Project Oversight Board (POB). The Computing Coordinator is a member of the LCG Project Execution Board (PEB). The Software Project Leader is a member of the LCG Applications Area Architects Forum (AA). ATLAS is also represented by the Computing Coordinator, the Computing Resources Manager and the Computing Operations Manager in the LCG Grid Deployment Board (GDB). Four ATLAS collaborators are members of the LCG Grid Applications Group (GAG). ATLAS also takes an active part in the SC2 (the LCG project monitoring body) and the LHC4-Computing group (the regular discussion group that includes the ATLAS spokesperson, the deputy in charge of computing, the computing coordinator, the ICB chair and the members of the other LHC experiments holding equivalent positions).

In order to ensure efficient cross-communication between ATLAS representatives in all these bodies, the "ATLAS-LCG Management Team", grouping all these people, meets weekly to exchange relevant information.

8.2 Work Plan, Schedule and Milestones

8.2.1 Planning Methodology

The deliverables of the Software & Computing project are determined by the need to prepare a complete and coherent software suite and computing environment for the beginning of ATLAS data taking in 2007. At the same time, the project has to provide continued support for detector commissioning activities and for physics studies.

The continued developments of core as well as of algorithmic code and of the computing environment prompted us to organize a series of so-called Data Challenges, starting in 2001. The scope of the Data Challenges is the test of increasingly complete software and infrastructure at progressively larger scales:

- DC0 in 2001 proved that we could simulate and reconstruct a sizeable number of events (10^5) with the same software release.
- DC1 in 2002–2003 proved that we could simulate, reconstruct and store in a distributed way ~15M events (still generated with Geant3 and reconstructed in the old FORTRAN-based framework).
- DC2 in 2004–2005 proved that we could use the Grid to generate ~15M events using Geant4, and reconstruct them with the new C++ algorithms in the Athena framework. Later in 2005 the distributed analysis system will be tested on these, and other, data samples.
- DC3 (also called "Computing System Commissioning") in 2006 will test all components of the Computing Model that have not so far been integrated with the mainstream software developments, such as the simulation of the trigger chain and the calibration and alignment procedures (and the simulation of "imperfect" data).

Complementary tests to the Data Challenges are those carried out in the context of the LCG Service Challenges. While the main aim of the Service Challenges is to set up the world-wide infrastructure that will allow the operation of our Computing Model, it is important that our pro-

duction tools be integrated in those tests, so as to be tested extensively themselves at the same time.

Given these external constraints, it is the task of the Planning Officer, in consultation with the CMB and SPMB, to define the development and release plan for the necessary components. The development plan is then implemented in the Work Breakdown Structure (WBS) [8-1] and the details of the work packages are discussed with the appropriate groups. The WBS is revised approximately once per year.

Progress is followed through regular internal reports; currently the Project Progress Tracking (PPT) system [8-2], developed at CERN for the LHC accelerator and detector construction, is in use. The PPT system is clearly designed for construction projects, therefore we will have to evaluate in the near future how to adapt it (or which other system to use) to monitor continuous software development and optimization activities.

8.2.2 Data Challenge 3: Computing System Commissioning

In preparing for ATLAS physics data in 2007, two complementary planning activities are under way. One of these deals with the use of the offline software and computing system in support of the commissioning of the ATLAS detector sub-systems as described in Section 3.12, "Use of Offline Software for Detector Commissioning". The other focuses on the commissioning of the computing system itself, the goal being to ensure that it meets minimal functionality, performance and robustness requirements for experiment turn-on.

Detailed planning for the computing system commissioning is under way, with the following major tests being planned:

- Offline Software Chain
- Tier-0 Scaling
- Calibration and Alignment
- Full Trigger Chain including Monitoring
- Data Distribution
- Distributed Analysis
- Distributed Production (Simulation & Re-processing)
- Full TDAQ/Offline Chain.

The strategy is to allow, as much as possible, for these tests to take place in parallel, although some couplings obviously exist and will be factored into the detailed scheduling.

A set of detailed acceptance criteria is being established for each test and these will be incorporated into the recurrent testing and validation procedures described in Section 3.13 to ensure that these criteria continue to be met. They set functionality, technical performance, and physics performance thresholds that need to be met or exceeded. For example, acceptance criteria for the Full Software Chain will include:

- Validation that the output from each stage in the processing chain is readable by the next stage.
- Demonstration of reconstruction from byte-stream files as well as POOL files.

- Non-recoverable error rates.
- Event processing times.
- ESD and AOD file sizes.
- Job start-up time.
- Memory usage.

The TDAQ/Offline full chain test is the culmination of the commissioning process, corresponding to the software and computing systems being ready for ATLAS physics.

This commissioning activity is scheduled to start in autumn 2005, and continue through to ATLAS turn-on.

8.2.3 High-Level Milestones

The list of milestones is intimately connected to the WBS. Here we list only the high-level milestones, i.e. those that, if missed, would have an impact on the ability of the Software & Computing project to satisfy its users.

- July 2005: test the TAG and event-collection infrastructure for data access for analysis (important for the Computing Model and the Event Data Model).
- July 2005: decision on the future baseline for the Distributed Analysis System.
- September 2005: new version of the Production System fully tested and deployed.
- September 2005: delivery of software and infrastructure for ATLAS Commissioning and for Computing System Commissioning; start of integration testing.
- October 2005: review of the implementation of the Event Data Model for reconstruction.
- October 2005: Distributed Data Management system ready for operation with LCG Service Challenge 3.
- November 2005: start of simulation production for Computing System Commissioning.
- January 2006: production release for Computing System Commissioning and early cosmic-ray studies; completion of the implementation of the Event Data Model for reconstruction.
- February 2006: start of DC3 (Computing System Commissioning).
- April 2006: integration of ATLAS components with LCG Service Challenge 4.
- July 2006: production release for the main cosmic-ray runs (starting in autumn 2006).
- December 2006: production release for early real proton data.

8.3 Manpower and Hardware Resources

8.3.1 Manpower Resources

Up to, and including 2005, the totality of the manpower for the Software & Computing project has been made available through voluntary contributions of the funding agencies that are members of the ATLAS Collaboration. As is always the case with voluntary contributions, the effort tends to be concentrated on topics that are more intellectually interesting for the contributors and beneficial to the career development of the people concerned; other vital areas of software and infrastructure development and support are not covered adequately.

As this situation was common to all LHC experiments, in September 2003 the LHCC held a review of the manpower available, concentrating on "Core Software" [8-3]. The definition of "Core Software" used at that time included all developments relative to frameworks, databases, infrastructure and services; the review highlighted a considerable lack of manpower for these activities and resulted in a call for help to the Collaboration. Since then, the situation has improved considerably in the field of databases, thanks also to the organization of the ATLAS-wide Database and Data Management project.

One of the recurring problems of the Software & Computing project has been the lack of formal acknowledgement of the effort committed to it. As the development of the software tools and the computing environment is a necessary component of the construction and operation of the ATLAS detector, the Software & Computing project had to be included in a formal Memorandum of Understanding between the members of the Collaboration. The approval of the Computing Addendum to the ATLAS Maintenance and Operations Memorandum of Understanding ("M&O MoU") [8-4] by the Resource Review Board in April 2005 has remedied this situation.

As described in the Computing Addendum, the "Core Computing" part of the ATLAS Software & Computing project covered by the M&O MoU is concerned with the development and maintenance of the experiment software framework and databases, the web and other documentation, the software infrastructure, with visualization and with the production tools. It also includes ATLAS software distribution and the provision and support of its interfaces to the Grid and LCG software.

The Core Computing activity concerns computing after the online; it does not concern computing local to a detectors subsystem (including the DAQ), nor computing in the trigger system or event filter. The Core Computing M&O contribution provides common infrastructure to the detector systems, as well as to the rest of offline computing activities. While it uses some tools provided by CERN-IT and the LCG project, the work is experiment-specific, and hence not covered by those bodies.

The Core Computing is divided into two categories (A and B) because of a clear distinction in the nature of the tasks involved and the rewards in terms of professional development associated with the tasks.

Category A tasks are extremely technical in nature. They are in general service tasks with required skill sets that are not generally compatible with those of a physicist. They are also at a level whereby they take a significant fraction of a Full-Time Equivalent (FTE) for the people carrying them out. These tasks are, however, crucial to the functioning of the experiment. Several funding agencies are already providing effort in these 'unglamorous' areas. The M&O mecha-

nism ensures a contribution from all agencies without dizadvantaging those who are already contributing. The analogy is with detector construction, where technical effort is brought in to cover many tasks. The M&O Addendum provides such a budget for the infrastructure and services part of Core Computing.

Category B activities are also vital, but the tasks involved have more obvious rewards in terms of status for the physicists fulfilling them, and for their funding agencies.

Both categories of effort are monitored through the project organization, with a resources officer comparing the work reported on a quarterly basis with a defined Work Breakdown Structure. This process has been in place for several years; the allocation of resources and resulting milestones have been systematically monitored and reported. It is this mechanism that has identified the persistent and severe shortfall in category A tasks.

8.3.1.1 M&O Category A

The Core Computing project provides a number of services to the entire Collaboration; the cost of these services has to be shared by the Collaboration as contributions to the Category A M&O budget.

Services needed by the Collaboration are listed in Table 8-1, together with the required manpower levels for 2005–2006. So far it has proven impossible to provide the manpower to cover the needed tasks through voluntary contributions only. It is hoped that the inclusion of these tasks in the Category A budget will encourage in-kind (manpower) contributions from the Funding Agencies, thus reducing the need to finance these activities centrally.

As the work items listed in Table 8-1 consist of services to the Collaboration, it is expected that if the required effort is delivered on time, the required manpower in subsequent years will not increase. The present estimate is that the resource needs will continue at the 2006 level up to 2010. A more detailed profile will be provided in 2007, based on the progress made in 2005 and 2006. Even though some tasks will become more efficient and require less effort (such as quality assurance framework development), other activities (such as user support and database services) will grow as the software moves from the hands of the developers to those of the users.

The current planning of core computing (Cat. A) activities assumes that in 2005 the Funding Agencies will make 10.3 FTEs available for computing infrastructure and service-related activities. These people are currently providing partial cover for all tasks listed in Table 8-1. The required effort in 2006 of 16.4 FTEs includes the projected 10 FTEs made available as continuations of existing in-kind contributions. Priorities are set in order to ensure the availability of the computing infrastructure in time for the physics start-up in July 2007. Should the 16.4 FTEs not be available for one reason or another, the level of user services will be most affected, thus resulting in delays in the physics analysis chain.

Table 8-1 Description of work items for work packages covered by the M&O-A budget for 2006

WBS items		Description	FTEs needed (2006)	Tasks and job profiles
1.2.9 Infrastructure & Services	1.2.9.1 Central Computing Environment	1.2.9.1.1 Standard Operating System and Grid Environment	**0.2**	Support for the ATLAS environment on default platforms.
		1.2.9.1.2 Dedicated developer platforms and general desktop support and operations	**0.2**	Support for the ATLAS environment on other common development platforms.
		1.2.9.1.3 Database administration	**1.0**	Administration of central ATLAS database servers.
		1.2.9.1.3 Platform and compiler validation and support	**0.7**	Porting ATLAS code to future platforms and compilers.
		1.2.9.1.4 WWW server, collaboration DB, document storage	**0.3**	Support for the ATLAS computing web and wiki.
		1.2.9.1.5 Support of videoconferencing, email lists, news, calendar, agendas	**0.2**	Setting up collaborative tools to support remote participation in meetings and discussions.
	1.2.9.2 User support	1.2.9.2.1 Help-desk system and staff	**1.5**	Help desk. Bug reporting system. Help mailing list.
		1.2.9.2.2 User and resource administration at CERN and Grid virtual organisation	**1.2**	Create and manage accounts and allocate disk space at CERN. Manage the ATLAS Virtual Organisation on the Grid.
		1.2.9.2.3 Computing and software documentation and training	**1.0**	Support for documentation in doxygen and html. Workbook. Organisation of tutorials.
	1.2.9.3 Software Process Services	1.2.9.3.1 Software repository management, access rights and mirroring	**1.0**	CVS repository support. Librarian.
		1.2.9.3.2 Configuration management, tag collection, software releases, nightly builds	**3.0**	CMT support, tag collector system, nightly builds, developer and production release builds.
		1.2.9.3.3 Software packaging and offsite installation and testing	**0.6**	Building distribution kits, installation on the Grid and validation testing.
		1.2.9.3.4 Software quality: code/dependency checking, regression/integration testing, bug-tracking, performance	**1.5**	QA/QC testing, testing framework, RTT system, valgrind.
1.4.2 Central Productions	1.4.2.2 Central Production Operations	1.4.2.2.2 Central Support for Production Tools	**1.0**	Deployment and support of production tools for Tier-0 and centrally managed remote productions.
		1.4.2.2.3 Central production operations, request and resource management	**2.0**	Management of Tier-0 and centrally managed remote productions.
		1.4.2.2.4 Integrating productions including error correction and catalogue publishing	**1.0**	Validation and checking of Tier-0 and centrally managed remote productions.
Total			**16.4**	

8.3.1.2 M&O Category B

The bulk of the Core Computing effort is based on voluntary, but recognized, manpower contributions, and as such can be treated in the M&O-B framework. For information, we give in Table 8-2 the list of activities and current (2005) manpower levels.

It should be noted that the available manpower has been below 70% of the requirement in almost all software development sectors for several years. This situation has generated delays in the preparation of the software infrastructure and framework, as well as in the development of algorithmic code. In order to catch up and be ready for detector commissioning and experiment turn-on, a much-increased participation in Core Computing Cat. B activities for 2006 and be-

yond is necessary. The inclusion of the tasks into the M&O process allows due credit to be given for the voluntary contributions made.

Table 8-2 Activities covered by M&O Category B and current (2005) manpower levels.

WBS	Activity		FTEs in 2005
1.1	Computing Coordination and Management		3.2
1.2	Software Project		
1.2.1		Coordination & Management	3.0
1.2.7		Simulation: Generators, Geant4 framework, Digitization, fast simulation	6.3
1.2.8		Core Services: Athena framework, Databases, Geometry, EDM, Graphics	10.5
1.2.10		Event Selection, Reconstruction and Analysis Tools	4.3
1.3	Database & Data Management (all activities, including non-offline)		19.5
1.4	Computing Operations		
1.4.1		Grid, Data Challenges & World-wide operations	15.0
1.4.3		Grid Tools and Services development and deployment	12.0
1.4.4		Operations Management	2.3
	Total		**76.1**

8.3.1.3 Other Software Developments

The development of software for detector and trigger studies, and of algorithmic code for calibration, reconstruction and analysis is not covered by the M&O MoU, as it is thought that these activities fit naturally into what is expected from all physicists who are members of the ATLAS Collaboration. These activities are nevertheless necessary, therefore they are coordinated by the Software & Computing project and acknowledged internally by the Collaboration.

Approximately 90 FTEs are contributing in 2005 to detector software developments (30 each to the Inner Detector and Muon System, 25 to the Liquid Argon Calorimeters and 5 to the Tile Calorimeter). This level of manpower may seem large but has to be compared with the large number of tasks still outstanding in the WBS [8-1]. Our partial success in attracting people to work on core software during last year resulted in a decrease in the number of available expert developers of detector software.

8.3.2 Hardware Resources

All hardware resources necessary for the operation of the ATLAS Computing Model will be provided by the LCG Collaboration. The LCG Memorandum of Understanding [8-5] defines the collaboration framework and the contributed resources (hardware and service levels). In this document we define only the resource needs as a function of time (see Chapter 7).

A small amount of hardware is required for central build and server machines. These are not covered in the online M&O for ATLAS, being primarily for offline code development and distribution, and for offline documentation services. Hardware (CPUs and disk) replacements and

extensions are needed for the build system that is currently hosted by the CERN computer centre. A number of machines need to be configured differently from those offered by the IT Department services, and have to be under the control of ATLAS. This is in order to test new platforms, operating systems and compilers, and to build software to be used by collaborating institutions that have different infrastructure from that provided by CERN-IT. The yearly requests for this hardware are submitted to the RRB as part of the M&O Category A contributions.

8.4 References

8-1 The ATLAS Software & Computing WBS can be found in:
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/planning

8-2 PPT project:
http://ppt.cern.ch

8-3 *LHCC Review of Computing Manpower*, CERN-LHCC-2003-036 / G-045

8-4 *Computing Addendum to the ATLAS Maintenance and Operations Memorandum of Understanding*, CERN-RRB-2005-008

8-5 *LCG Memorandum of Understanding*, CERN-CRRB-2005-001

A Glossary

A.1 Acronyms and Terms

ADA	ATLAS Distributed Analysis system
AFS	Andrew File System
AMI	ATLAS Metadata Interface
ARDA	Architectural Roadmap towards Distributed Analysis (Grid)
AOD	Analysis Object Data
API	Application Program Interface
ARC	Advanced Resource Connector (Grid)
ATLAS	A Toroidal LHC ApparatuS
Athena	ATLAS offline software framework
BDII	Berkeley Database Information Index (LCG GIIS replacement)
C++	Primary programming language used in ATLAS
CA	Certification Authority (Grid)
CASTOR	CERN Advanced STORage Manager
CBNT	Combined N-tuples (Software)
CE	Computing Element (Grid)
CERN	European Laboratory for Particle Physics
CINT	C/C++ interpreter that is embedded in ROOT
CLHEP	Class Library for HEP (Software)
CMT	Configuration Management Tool
CondDB	Conditions Database
CondorG	Grid Batch System
COOL	LCG Conditions Database Project
CTB	Combined Test Beam
CVS	Concurrent Versioning System
DAQ	Data Acquisition System
DBMS	Database Management System
DC	Data Challenge (Operations)
dCache	Disk Cache System
DCS	Detector Control System (Software/Data Management)
DIAL	Distributed Interactive Analysis of Large datasets
DM	Data Management

DN	Distinguished Name (Grid certificate)
DPD	Derived Physics Data (Software)
DQ	DonQuijote, distributed data manager (Data Management/Grid)
DRD	Derived Reconstruction Data (Computing Model)
EB	Event Builder
EDM	Event Data Model
EDG	European Data Grid project
EF	Event Filter
EGEE	Enabling Grid for E-sciencE project
ESD	Event Summary Data
FC	File catalogue
FORTRAN	Programming language
FTP	File Transfer Protocol
GACL	Grid Access Control List
GANGA	Gaudi/Athena Grid user interface
Gaudi	LHCb Data Processing Applications Framework
GEANT	Pan-European Gigabit Research and Education Network
Geant4	A toolkit for simulation of the passage of particles through matter
GID	Global Event Identifier
gLite	Lightweight middleware for Grid Computing
Globus	Grid Middleware
GLUE	Grid Laboratory Uniform Environment
Grid3	US Grid System
GridFTP	Protocol Extensions to FTP for the Grid
GSI	Grid Security Infrastructure
GUI	Graphical User Interface
GUID	Globally Unique Identifier (Grid)
HBOOK	Legacy histogramming package
HepMC	MC Generator 4-Vector Classes
HEP	High-energy physics
HLT	High-Level Trigger
HSM	Hierarchical Storage Manager
HTML	Hypertext Mark-up Language
HVS	Hierarchical Versioning System (Software)
IDC	Identifiable Container

IOV	Interval of Validity
IPC	Inter-Process Communication
Java	Programming language
LAN	Local Area Network
LCG	LHC Computing Grid
LDAP	Lightweight Directory Access Protocol
LFN	Logical File Name
LHC	Large Hadron Collider
LSF	Load Sharing Facility (Batch system)
M&O	Maintenance and Operation
MC	Monte Carlo simulation
MDS	Monitoring and Directory Service (Grid)
MONARC	Models Of Networked Analysis at Regional Centres
MoU	Memorandum of Understanding
MSSM	Minimal SuperSymetric Model
MTTF	Mean Time To Failure
NOVA	Networked Object-based Environment for Analysis (Data Management)
OLE	Object Linking and Embedding
OO	Object Oriented (Software)
OS	Operating System
OSG	Open Science Grid (US)
Pacman	Package Manager
PASTA	LCG technology tracking team
PAW	Physics Analysis Workstation (Legacy data analysis package)
PBS	Portable Batch System
PESA	Physics and Event Selection Architecture
POOL	Pool Of persistent Objects for LHC (Data Management)
PROOF	Parallel ROOT Facility
Python	Interpreted programming language
RAL	Relational Access Layer (Data Management)
RAW	Raw Data
RB	Resource Broker (Grid)
RC	Replica Catalogue (Grid)
RDBMS	Relation Database Management System
RDO	Raw Data Object

RFIO	Remote File I/O
RHEL	Redhat Enterprise Linux
RLS	Replica Location Service (Grid)
RIO	Reconstruction Input Object
ROB	Read-Out Buffer
ROD	Read-Out Driver
RoI	Region of Interest
ROOT	A class library for data analysis.
RSL	Resource Specification Language (Grid)
RTAG	Requirements Technical Assessment Group (LCG)
SDO	Simulation Data Objects
SE	Storage Element (Grid)
SEAL	Shared Environment for Applications at LHC
SG	StoreGate (Software)
SGE	Sun Grid Engine (Batch Job System)
SLC3	Scientific Linux CERN 3
SI2k	SpecInt2000 (CPU Benchmark)
SIM	Simulated Data (RAW Format)
SIT	Software Infrastructure Team (of ATLAS)
SRM	Storage Resource Manager (Grid)
STL	Standard Template Library (C++)
TAG	Event-level metadata
TCP/IP	Transmission Control Protocol/Internet Protocol
TDAQ	ATLAS Trigger and DAQ (including DCS)
TDR	Technical Design Report
TDS	Transient Data Store
TES	Transient Event Store
TOF	Time Of Flight
TURL	Transfer URL (Grid)
UI	User Interface
URL	Universal Resource Locator (Grid)
VO	Virtual Organization (Grid)
VOMS	Virtual Organization Membership Service (Grid)
WAN	Wide Area Network
WMS	Workload Management System (Grid)

WN	Worker Node (Grid)
XML	Extensible Markup Language
ZEBRA	Legacy data management system

A.2 Definitions

A

AOD

The Analysis Object Data, which consists of a reduced size output of physics quantities from the reconstruction that should suffice for most kinds of analysis work.

See also ESD and Section 3.9.3.1.

Athena

The software framework for ATLAS. Based on Gaudi, Athena provides common services such as the transient data store, (interactive) job configuration and auditing, data(-base) access, message streams, etc. for ATLAS software. The idea is to improve the coherency of the different software domains within ATLAS, and thereby the ease of use for end-users and developers, by having them all use the same well-defined interfaces.

See also Gaudi and Section 3.3

Atlfast

The ATLAS fast simulation program (Atlfast) simulates ATLAS physics events, including the effects due to detector response and the software reconstruction chain. The input to the program is the collection of four-vectors for a physics event, usually provided by a physics event generator. Smearing and jet finding algorithms are applied to the energy deposits, and the resulting jet objects are output for further physics analysis.

See also Section 3.8.3

C

Certificate

A public-key certificate is a digitally signed statement from one entity (e.g., a certificate authority), saying that the public key (and some other information) of another entity (e.g., the Grid user) has some specific value. The X.509 standard defines what information can go into a certificate, and describes how to write it down (the data format).

CMT

A Configuration Management Tool: it is used for building software releases, for package dependency management, and for the setup of the run-time environment. Practically everything is treated as a package by CMT, even the run-time environment which is setup by "compiling" a run-time package. Much of CMT's behaviour (and hence how it should be used and configured) is determined by project-specific policy files.

Compute Element (CE)

Compute element (also called a compute service or CS) is a term used in Grids to denote any kind of computing interface, e.g., a job entry or batch system. A compute element consists of one or more similar machines, managed by a single scheduler/job queue, which is set up to accept and run Grid jobs. The machines do not need to be identical, but must have the same OS and the same processor architecture. A CE must run (among other things) a process called the "gatekeeper".

Conditions Database (CondDB)

The conditions database contains the record of the detector conditions required for data analysis, e.g. calibration and geometry constants.

See also Section 4.4, "Conditions and Configuration Data"

CVS

A concurrent versions system that allows for sharing of source code among the members of a distributed development team, such as ATLAS. All offline code resides in the official repository, which can be browsed online and from which users can download ("check-out") code.

See also CMT.

D

Data Acquisition System (DAQ)

The detector system comprising the Data Flow, Online and Detector Control Systems.

Data Challenge (DC)

A data challenge comprises, in essence, the simulation, done as realistically as possible, of data (events) from the detector, followed by the processing of those data using the software and computing infrastructure that will, with further development, be used for the real data when the LHC starts operating. The goals of the ATLAS Data Challenges are the validation of the ATLAS Computing Model, of the complete software suite, of the data model, and to ensure the correctness of the technical computing choices to be made. In addition the data produced allowed physicists to perform studies of the detector and of different physics channels.

See also Section 6.5, " Experience with Data Challenges and other Mass Productions".

Dictionary

Also known as reflection information, a dictionary refers to an object that contains a detailed description of another object. This information includes the object type, member functions and arguments, member data, etc. There are two prevailing dictionaries in HEP: the one provided by SEAL (typically referred to as the LCG dictionary) and the one provided by ROOT.

See also CINT, LCG, SEAL, ROOT.

Distribution Kit

The ATLAS software is distributable through a set of tools, collectively referred to as the "Distribution Kit." Basically, it contains all the binaries and header files for a specific platform. The distribution kit can be downloaded from a website and installed with Pacman.

See also Pacman and Section 3.14.6, "Code Distribution".

E

Event Filter (EF)

Sub-system of HLT comprising the hardware and software for the final stage of the online event selection, data monitoring and calibration using offline style algorithms operating on complete events accepted by LVL2.

ESD

The Event Summary Data, which consists of sufficient information to re-run (parts of) the reconstruction. For some kinds of analysis, the information available in the AOD may not be enough. By selecting information from the ESD (or by selecting certain reconstruction algorithms and hence, transparently, the required ESD), rather than going back to the original byte stream, the needed I/O can still be kept to a minimum.

See also AOD and Section 3.9.2.

G

Gaudi

The software framework for LHCb. The Gaudi framework was originally developed by LHCb, but is now in use by several experiments, including ATLAS. When referring to Gaudi in the context of Athena, the term is meant to describe the common core of the two frameworks. For more information, see the Gaudi project site.

See also Athena.

Geant4

A toolkit for building geometries of physical structures, e.g. the ATLAS detector, and simulating the physics processes that occur as particles traverse through these structures. For use within ATLAS, see Section 3.8.4.

Generator

Short for "event generator." A computer program that models the physics processes that occur in the collisions in high-energy physics experiments. The results of running a generator consist of a list of particles (incoming, created, decay products, etc.), their properties, and their origins, just after the collision.

See also Pythia.

GeoModel

The GeoModel toolkit is a library of geometrical primitives. The toolkit is designed as a data layer, and especially optimized in order to be able to describe large and complex detector systems with minimum memory consumption. GeoModel is used for the description of the ATLAS detector geometry. A faithful representation of a GeoModel description can be transferred to Geant4.

See also Section 3.5.2

Grid

An infrastructure of computing resources, such as storage and processing time, that is transparently made available to the user through a network.

H

HepMC

An event record for high-energy physics Monte Carlo generators. HepMC stores the output from an event generator as a graph of particles and vertices, where the vertices maintain a listings of the incoming and outgoing particles and the particles point back to their production vertices.

High-Level Triggers (HLT)

Trigger/DAQ system, comprised of both the LVL2 and EF, the two ATLAS trigger levels that are implemented primarily in software.

J

Job Options

In the Athena framework a job is specified in terms of a set of job options (as Python statements). Via job options the dynamic libraries to load are specified, the algorithms are selected and the sequence of execution and the properties of the algorithms are defined.

L

LCG

The LHC Computing Grid Project, which intends to serve the computing needs of LHC by deploying a world-wide Grid, integrating the capacity of scientific computing centres spread across Europe, America and Asia into a virtual computing organization.

See also Grid, POOL, SEAL.

Level-1 Trigger (LVL1)

The ATLAS First Level Trigger system provides a hardware based first trigger decision using only a sub-set of an event's data (Calorimeter and Muon only). Normally, only the events accepted by LVL1 are transferred from the detectors into the HLT/DAQ system.

Level-2 Trigger (LVL2)

Sub-system of HLT which provides a software based second stage trigger decision, to reduce the rate of triggers from LVL1 by about a factor of 100. It uses 'Regions of Interest' (RoIs) as given by the LVL1 trigger to selectively read out only certain parts of the ATLAS detector hardware and computes a LVL2 trigger decision.

LSF

The Load Sharing Facility is a batch queuing system that regulates the use of computing resources. Jobs are submitted into queues based on their expected load and executed in an as fair as possible scheme following pre-set priorities. Detailed information and reference guides are available at the CERN batch services site.

M

Middleware

Middleware is software that connects two or more otherwise separate applications across the Internet or local area networks. More specifically, the term refers to an evolving layer of services that resides between the network and more traditional applications for managing security, access and information exchange.

P

Pacman

A PACkage MANager, which allows one to install parts of, or the whole, ATLAS offline software release on a local machine.

See also Distribution Kit.

POOL

A Pool Of persistent Objects for LHC, POOL is a persistency framework designed for the LHC experiments and allows for the storage of the large volumes of experiment data and metadata in

a Grid enabled way. Normally, within the Athena framework, one does not deal directly with POOL. Instead, POOL is specified as a conversion service (that is, the facility that converts the transient data into persistent data) in the configuration of the job and, unless one has special needs, its use is transparent.

See also Athena, LCG and Section 3.15.2, Section 4.5.

Pythia

A program for the generation of high-energy physics events. The simulation of events as they will take place in the ATLAS detector, starts with the simulation of the collisions themselves. This is done in event generators such as Pythia, which contains theory and models for e.g. hard and soft interactions, parton distributions, initial and final state parton showers, particle decay, etc. The output of an event generator can subsequently be fed into the ATLAS detector simulation, which tracks the particles from the events through ATLAS. Detailed information is available on the Pythia site. There is also ATLAS specific information, which describes the Pythia_i interface package for use with Athena.

Python

An interpreted, interactive, object-oriented, open-source programming language. The Python interpreter is used as the scripting interface for Athena, because of its excellent extending and embedding facilities.

ROOT

A class library for data analysis. The ROOT package provides an extensive set of functionality for data analysis in high-energy physics; it includes data display, persistency, minimization, fundamental classes, etc., as well as an interactive interpreter. The full class reference, tutorials, and other documentation, are available at the ROOT project site.

S

SEAL

The Shared Environment for Applications at LHC, which provides common core and services (such as system, utility, framework, and mathematics) libraries for LCG applications in order to improve their coherency. For the ATLAS end-user, the most visible products of SEAL are the LCG dictionary and the scripting services such as PyROOT and PyBus.

See also LCG and Section 3.15.1.

SRM

Grid middleware component Storage Resource Manager, used for data management and virtualization of storage interfaces. SRM provides shared storage resource allocation and scheduling. It manages space, manages files on behalf of users, manages file sharing, manages multi-file requests, and provides Grid access to and from a mass storage system. An SRM does not perform file transfers, rather it invokes file transfer services as needed, monitors transfers and recovers from failures.

Storage Element (SE)

Any data storage resource that is registered in a Grid Information Service (GIS), contains files registered in a Replica Location Service (RLS), and provides access to remote sites via a Grid interface (e.g., GSI authenticated). A Storage Element (SE) provides uniform access and services to large storage spaces. The storage element may control large disk arrays, mass storage systems and the like.

StoreGate

StoreGate implements a transient data store for Athena and is described in the Athena manual. For more information, see Section 3.4.2.

See also Athena, TDS.

T

TAG

Used in many contexts, here we mean event-level metadata, i.e. global event quantities with reference to the event AOD. TAG databases are used for fast event selection and creation of event sample lists, see Section 4.5.8, "Event-level Metadata".

TDS

The Transient Data Store (TDS) is a box where data producers can drop off their results to be picked up by clients (more often referred to as consumers, even though they don't actually consume the data). The use of a store decouples producers from clients. Examples of producers include algorithms that perform calculations to come up with new data, but also services that read data from disk. Clients can be other algorithms that perform further calculations, or services that write data to disk or a database. In Athena, the transient store is implemented by StoreGate.

See also Athena, StoreGate.

Tier-0

Initial Tier in the Grid hierarchy; it is the site at which raw data is taken. The experimental online system interfaces to the Tier-0 resources. For the LHC experiments, CERN is the Tier-0 facility.

Tier-1

Next Tier, after Tier-0, in Grid hierarchy. Tier-1 sites are connected to a Tier-0 site based on an MoU with the Tier-0 site. Typically a Tier-1 site offers storage, analysis, and services, and represents a broad constituency (e.g., there may be a single Tier-1 site per country or region which connects with multiple Tier-2 sites in that country or region). For ATLAS, 10 Tier-1 sites are planned.

Tier-2

Tier-2 is the next level down in the Grid hierarchy of sites, after Tier-1. Tier-2 sites are typically regional computing facilities at University institutions providing a distributed Grid of facilities.

V

Virtual Organization (VO)

A participating organization in a Grid to which Grid end users must be registered and authenticated in order to gain access to the Grid's resources. A VO must establish resource-usage agreements with Grid resource providers. Members of a VO may come from many different home institutions, may have in common only a general interest or goal (e.g., ATLAS physics analysis), and may communicate and coordinate their work solely through information technology (hence the term virtual). An organization like an HEP experiment can be regarded as one VO.