Development of the ATLAS Software and Simulation of the b-quark Physics on ATLAS

Julius Hřivnáč

Contents

Acknowledgment

The author of these theses thanks

to all the *Moose* friends for helping him to discover the exciting world of the modern software,

to all colleagues from the *PAtlas FST* group for a very fruitful collaboration on the Atlas topics,

to all colleagues from the *ATLAS Graphics* Team for explaining him the graphics secrets,

to his wife and daughter for the patience during all that time he was working on the text.

Chapter 1

Introduction

This theses cover two fields of the preparation of the ATLAS [113] experiment in CERN [3]: the development of the software system and the physics of the B-quark. Naturally, only subsets of all topics are covered. In the software part, the Software Process and the development of the Graphics software are described. A note about the importance of Java to ATLAS is added in the Appendix. The physics part contains the study about the CP Asymmetry Dilution Factor dependency on the ATLAS detector setup and the study of the Beauty Baryon Polarization. The original work of the author is contained mostly in Chapters 4, 5 and 6 and Appendix B.

Chapter 2 contains a short rationale and a description of the LHC [4] machine being built in CERN.

Chapter 3 provides an overview description of the ATLAS detector. Physics requirements for all its parts as well as their main characteristics are summarized.

Chapter 4 deals with software issues in general. It describes relevant Programming Methodologies, Software Processes, Tools and Programming Languages of interest to the High Energy Physics. Most of the chapter is just compilation of various topics, some of which were, however, not available in such a distiled form to the HEP community until now. The most important (and to large extend original) contribution of the author is a set of general Software Design Rules for the Object Oriented Software Development (Section 4.1.2). Also guidelines for applying Extreme Programming Process (Section 4.2.7) are explained for the first time in the HEP community.

Chapter 5 concerns the ATLAS software. After a short description of the ATLAS software status (with a short analysis of the situation), three major software topics are covered. Section 5.3.2 describes original contribution to the ATLAS Software Process - The Handling guidelines for the External Software. The main original software contribution is described in the Section 5.4 - it covers the development of the ATLAS Graphics Systems. All relevant phases of the development are described: From the accumulated User Requirements (summarized in the Appendix B), through the top level Graphics Architecture to the Design of the key graphics component - the Graphics Control. The author of the thesis has coordinated the development of the Graphics Framework, has defined and developed its Architecture and has developed some of its major components (the most important is the overall Graphics Control Structure). All graphical components (Scenes) are described and the emphasis is laid on the Scene developed by the author of the thesis - the 3D Event Display GraXML. Also author's contributions beyond ATLAS, into FreeHEP library and HEP Studio, are mentioned. This Graphics Framework currently constitutes the baseline of the ATLAS Graphics environment and is integrated into the overall ATLAS Framework - Athena. The ATLAS Graphics system has been presented in several international conferences. Section 5.5 shortly mentions the original algorithm for the fast Pattern Recognition developed by the author, the implementation of it, however, hasn't gone beyond a prototype level.

Chapter 6 covers the Physics topics. After a short introduction to the ATLAS Physics and B-quark Physics in general, two topics are brought in. Section 6.3 describes the work done for the Evaluation of the CP Asymmetry Dilution Factor dependency on the ATLAS detector setup. While this topic doesn't bring any new physics theme, the results of it have been used in the formation of the decision between the various design choices for the ATLAS detector setup. Section 6.4 contains original contribution about the possibility of beauty baryon polarization measurement in ATLAS. The gained results have been published [92] and are also incorporated in the ATLAS Detector and Physics Technical Design Report [14]. The results in this Theses are updated with respect to published results to take into account new experimental data. Author has performed all the study described in Sections 6.3 and 6.4 and the simulation work described in general part of the Chapter in collaboration with his colleagues from the Prague ATLAS group, the Section 6.3 together with M.Smižanská, the Section 6.4 together with M.Smižanská and R.Lednický.

Chapter 7 gives the Conclusions.

Appendix A contains the the copy of the ATLAS internal Note about the role of Java in the ATLAS software, which has been co-written by author (with S.Fisher). Author has written Sections A.3 - A.8. Besides being an important contribution to the ATLAS software strategy making, its conclusions are highly relevant to the Software part of the Thesis.

Appendix B contains the list of the accumulated User Requirements for the Graphics Domain, which are described in the Section 5.4. All those requirements were accumulated by the author from his own experience and discussions with domain experts.

Chapter 2

LHC

This Chapter contains a short rationale and a description of the LHC machine being built in CERN.

In November 2000, the e^+e^- collider LEP [1] in CERN [3] has been definitely shut down. It has achieved about 209 GeV collision energy and further increase is not feasible in circular accelerator due to massive synchrotron radiation loss with the energy.

The large mass of hadrons with respect to electrons has the advantage that less energy is lost due to the bremsstrahlung effect. The main disadvantage of a hadron colliders is the compositeness of the hadron. This means that the available energy of the hadron is distributed over the individual constituents (quarks and gluons).

Taking into account the above arguments and as a new collider in CERN has to match existing LEP tunnel, the Large Hadron Collider - LHC [4] (see Fig. 2.1) has been designed. The LHC is currently being build in the existing LEP tunnel. It will be able to collide beams of 7 TeV protons in the first stage, Pb beams later. The *pp* luminosity will be 1.7 10^{34} $\mathrm{cm}^{-2}\mathrm{s}^{-1}$ with the bunch crossing time of 25 ns. For the inelastic cross section of 80 mb this leads to an average pileup of about 23 overlapping processes per beam crossing. The cross section for interesting processes (Higgs production, $t\bar{t}$ production,...) will be several orders of magnitude bellow the total cross section. This means that a large data rejection power will be needed.

The LHC experiments will be bigger and more complex than those at LEP and HERA, in detector size and complexity, number of physicists, event size, bunch crossing rate and experiment lifetime.

LHC is supposed to start its operation in 2005 and deliver full pp luminosity in 2007.

A Toroidal LHC ApparatuS - ATLAS [113] is one of the general-purpose detectors build in the LHC ring.

Compact Muon Solenoid - CMS [5] is the other one.
A Large Ion Collider Experiment - ALICE [6] is specialized on heavy ion physics (Pb-Pb collisions).
The Large Hadron Collider beauty Experiment - LHCb [7] is a dedicated B-physics experiment.

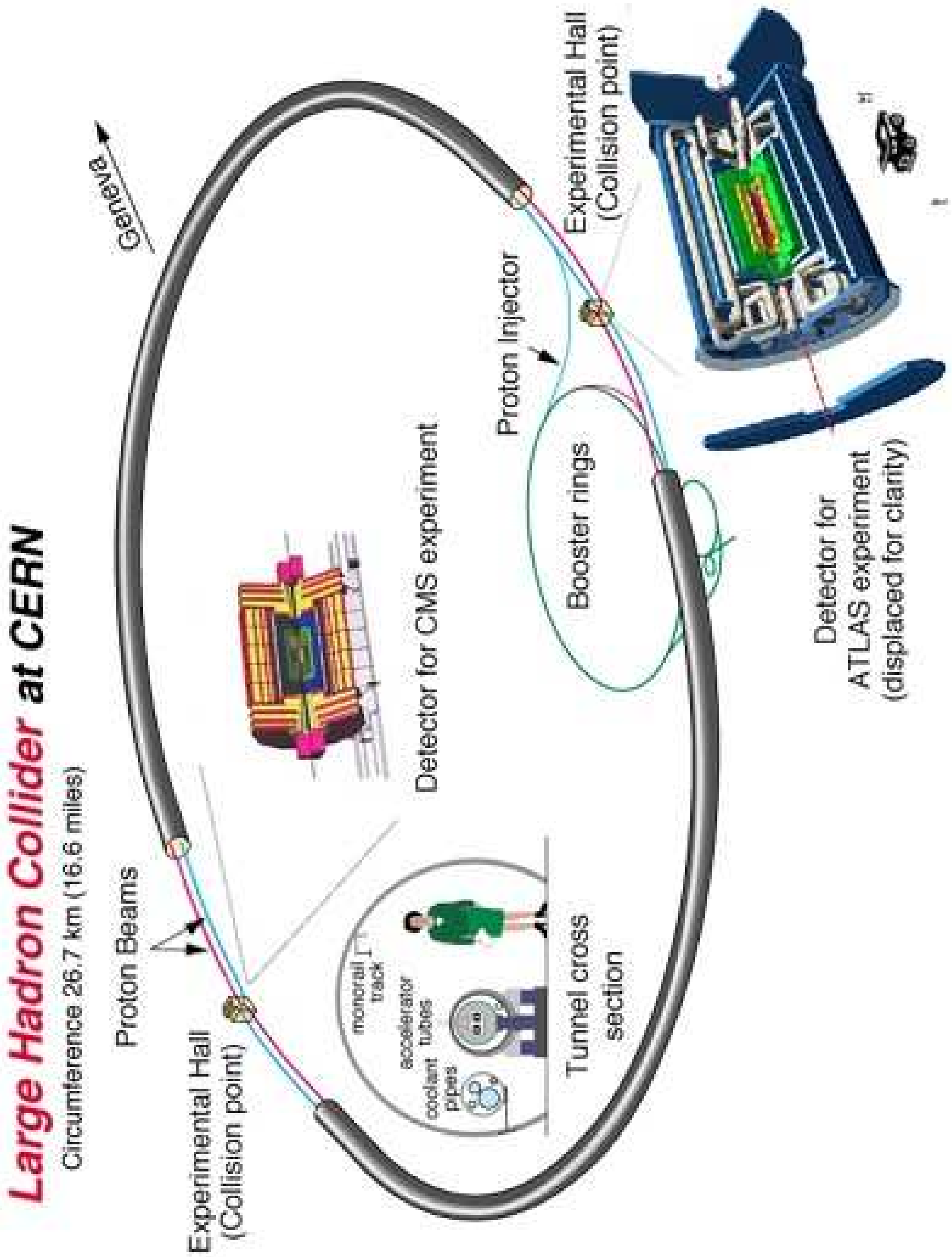

Figure 2.1: The Large Hadron Collider in CERN.

Chapter 3

ATLAS

This Chapter provides an overview description of the ATLAS detector. Physics requirements for all its parts as well as their main characteristics are summarized.

The ATLAS detector (shown in Fig. 3.1 and 3.2) is described in detail in the experiment's Technical Proposal [12] and Technical Design Reports [13] of all its components. Here, just the basic overview of the detector is given.

The primary goal of the experiment is to operate at high luminosity 10^{34} $\mathrm{cm}^{-2}\mathrm{s}^{-1}$ with a detector that provides as many signatures as possible. The variety of signatures is considered to be important in the harsh environment of the LHC in order to achieve robust and redundant physics measurements with the ability to cross-check.

3.1 Magnets

ATLAS contains two magnetic systems: an inner thin superconducting solenoid surrounding the inner detector cavity, and large superconducting air-core toroids consisting of independent coils arranged with an eight-fold symmetry outside the calorimeters.

3.2 Inner Detector

The Inner Detector - ID (shown in Fig. 3.3 and 3.4) is contained within a cylinder of a length 7 m and a radius of 1.15 m, in a solenoidal magnetic field of 2 T. Momentum and vertex measurements of particle tracks, as well as electron identification are achieved using pattern recognition techniques on the combined signals of discrete high-resolution semiconductor pixel and strip detectors in the inner part of the tracking volume and a continuous straw-tube tracking detectors with transition radiation capabilities in its outer part.

ID is designed to reconstruct tracks and decay vertices in any event with high efficiency. Important physics considerations for the inner detectors are:

- Excellent momentum and impact parameter resolution for tracks with p_T >0.5 GeV up to very high momentum.
- Tracking coverage over the range $|\eta| < 2.5$.
- High track finding efficiency whilst keeping the rate from fake tracks low.
- Identification of the charge of high-p_T tracks.
- Tagging of b-jets coming from b-quarks.
- Reconstruction of soft electrons and secondary vertices from b- and τ-decays.
- Identification of the primary vertex.
- Electron identification.

Tracks with $p_T < 0.5$ GeV loop in the magnetic field and cannot be reconstructed. The momentum resolution is limited by the strength of the magnetic field, the precision of the detector elements, the available space and for low p_T tracks by the amount of the material in ID.

ID contains a small number of high-precision layers in the *Pixel and Semiconducting Tracker (SCT)* and a large number of straw tubes in the *Transition Radiation Tracker (TRT).* The Pixel detector with 140 million channels gives resolution of about 10 μm in $r\phi$ and 50 μm in z. The four SCT double layers give a resolution of about 20 μm in $r\phi$ and 700 μm in z. Each track passes through more than 30 TRT straw tubes, giving possibility for a good pattern recognition and electron identification by their transition radiation.

The combined system gives a momentum resolution of about 20% for low-η tracks with 500 GeV of transverse momentum. At high momentum, the impact parameter of the track at the primary vertex can be measured with a precision of < 15 μm. At lower momenta, multiple scattering of the tracks reduces the precision to < 60 μm. These measurements allow 50% of b jets to be identified while reducing the background from light-quark jets by a factor > 70.

3.3 Calorimeters

Highly granular liquid-argon (LAr) electromagnetic (EM) sampling calorimetry, with excellent performance in terms of energy and position resolution, covers the pseudorapidity range $|\eta| < 3.2$. In the end-caps, the LAr technology is also used for the hadronic calorimetry, which share the cryostats with the EM end-caps. The same cryostats also house the special LAr forward calorimeters which extend the pseudorapidity coverage to $|\eta| = 4.9$. The bulk of the hadronic calorimetry is provided by a novel scintillator-tile calorimeter, which is separated into a large barrel and two smaller extended barrel cylinders, one on each side of the barrel. The overall calorimeter system provides the very good jet and E_T^{miss} performance of the detector.

The LAr calorimetry is contained in a cylinder with an outer radius of 2.25 m and extends longitudinally to ± 6.65 m along the beam axis. The outer radius of the scintillator-tile calorimeter is 4.25 m and its half length is 6.10 m. The total weight of the calorimeter system is about 4000 tons. (The overall weight of the ATLAS detector is about 7000 tons.) The ATLAS calorimeter system is shown on the Fig. 3.5.

The intrinsic energy resolution of calorimeters improves with increasing energy. Physics requirements on calorimeters at the LHC include:

- The accurate measurement of the energy and position of both electrons and photons.
- The measurement of the energy and direction of jets.
- Particle identification including the separation of electrons, photons and hadronic τ decays from jets.
- The measurement of the missing traverse momentum of the event.
- Event selection at the first trigger level.

The ATLAS calorimeters consist of an electromagnetic calorimeter with a barrel and two end-caps at inner radii covering a pseudorapidity range of $|\eta| < 3.2$, a barrel hadronic calorimeter covering $|\eta| < 1.7$, two end-cap hadronic calorimeters covering $1.5 < |\eta| < 3.2$ and forward calorimeters covering $3.1 < |\eta| < 4.9$.

The electromagnetic (EM) calorimeter reconstructs electrons and photons with energy from 2 GeV to 5 TeV. The energy resolution of 1% is required in this region. A rejection against hadronic jets of a factor of 5000 is needed in the range 25 GeV $< p_T <$ 100 GeV. The EM calorimeter is a lead LAr detector with accordion shaped Kapton electrodes and lead absorber plates. The total amount of material seen at the calorimeter face is about 2.3 radiation lengths at $\eta = 0$. A shower in EM calorimeter is shown on Fig. 3.6.

The main function of the hadronic calorimeter is to identify, measure and reconstruct jests and the missing p_T of an event. The ATLAS hadronic calorimeters cover the range of $|\eta| < 4.9$ using different techniques. Over the range of $|\eta| < 1.7$, the iron scintillating-tile is used for the barrel and extended barrel tile calorimeters. Over the range of about $1.5 < |\eta| < 4.9$, LAr calorimeters are used, hadronic end-cap calorimeter (HEC) and high-density forward calorimeter (FCAL). The total thickness at $\eta = 0$ is 11 interaction lengths.

3.4 Muon Spectrometer

The calorimeter is surrounded by the muon spectrometer, shown on Fig. 3.7 and 3.8. The air-core toroid system, with a long barrel and two inserted end-cap magnets, generates a large magnetic field volume with strong bending power within a light and open structure. Multiple-scattering effects are thereby minimized, and excellent muon momentum resolution is achieved with three stations of high-precision tracking chambers. The muon instrumentation also includes as a key component trigger chambers with very fast time response.

The muon spectrometer defines the overall dimensions of the ATLAS detector. The outer chambers of the barrel are at a radius of about 11 m. The half-length of the barrel toroid coils is 12.5 m and the third layer of the forward muon chambers is located at about 23 m from the interaction point.

The muon spectrometer is required to:

- The identification, reconstruction and measurement of the momenta of muon tracks, as well as the finding of an association with measurements in the inner detector. Good transverse momentum resolution must be maintained down to 5 GeV within pseudorapidity region of $|\eta| < 3$.

- The trigger on both single-muon and multi-muon events. The trigger system must have a wide acceptance, be efficient and selective.

- The association of all muons unambiguously to the correct bunch crossing using timing information provided by the trigger chambers.

A wide pseudorapidity coverage is essential for track reconstruction in rare processes. Momentum and mass resolution of the order of 1% are essential for the reconstruction of narrow final states with two or four muons, for background rejection and charge identification. The muon spectrometer must also measure a second coordinate in the non bending plane for muon tracks, with a spatial resolution of 5-10 mm to ensure good track reconstruction and momentum measurement. For the muon trigger system, transverse mass thresholds in the range 10-20 GeV are used for large mass particles, whereas the more stringent requirements come from B-physics, where a p_T threshold of down to 5 GeV is required. A trigger coverage of $|\eta| < 2.4$ is deemed to be adequate.

The ATLAS muon spectrometer consist of three large superconducting air core toroid magnets, a separate-function fast trigger system and high precisions tracking chambers. The magnet system provides the magnetic field within the pseudorapidity range of $|\eta| < 2.7$. The trigger chambers identify bunch crossings, trigger with well defined p_T thresholds and measure a second coordinate orthogonal to that measured in the muon chambers. In the barrel, the chambers are arranged on the three concentric cylinders at radii 5 m, 7.5 m and 10 m. In the end-caps, they are arranged over four disks at distances 7 m, 10 m, 14 m and 21 m. This arrangement is such that particles from the interaction point traverse three chambers.

3.5 Trigger

The ATLAS experiment will have to deal with the unprecedented bunch crossing rate of 40 MHz. There will be an average 23 interactions for each bunch crossing leading to an interaction rate of about 10^9 Hz. It is technically not possible to read out and store all these data and most of them even doesn't contain any interesting physics event. The online trigger system will have to be able to select interesting physics signatures and reduce the amount of data to approximately 100 MB/s. This requires the rejection power of about 10^7 against minimum bias processes as is shown (in more details) in Fig. 3.9).

The ATLAS Trigger system is organized in three levels: Level 1 (Lvl1), Level 2 (Lvl2) and Event Filter (EF) (see Fig. 3.10). The Trigger algorithms at Lvl1 act on reduced-granularity data from a subset of the detectors (the muon trigger chambers and the calorimeters) and are implemented in very fast custom hardware processors. Events selected by Lvl1 are read out from the front-end electronics systems of the detectors into *Read Out Drivers* (RODs) and then forwarded into *Read Out Buffers* (ROBs). All data for the selected bunch crossing are held in the ROBs until Lvl2 takes decision to either discard or to accept it. The Lvl2 uses full-granularity data from all detectors, but examines only regions identified by the Lvl1 as containing interesting information - *Regions of Interest* (RoIs). Accepted events are send via the event building system to the EF. The EF then uses the full data together with the latest available calibration and alignment information to make the final selection of events to be recorded for offline analysis.

The foreseen event size being read out from the Atlas detector is about 2.2 MB (447 kB for the Inner Detector, 1458 kB for Calorimeters, 175 kB for the Muon Spectrometer and 100 kB of the Trigger information).

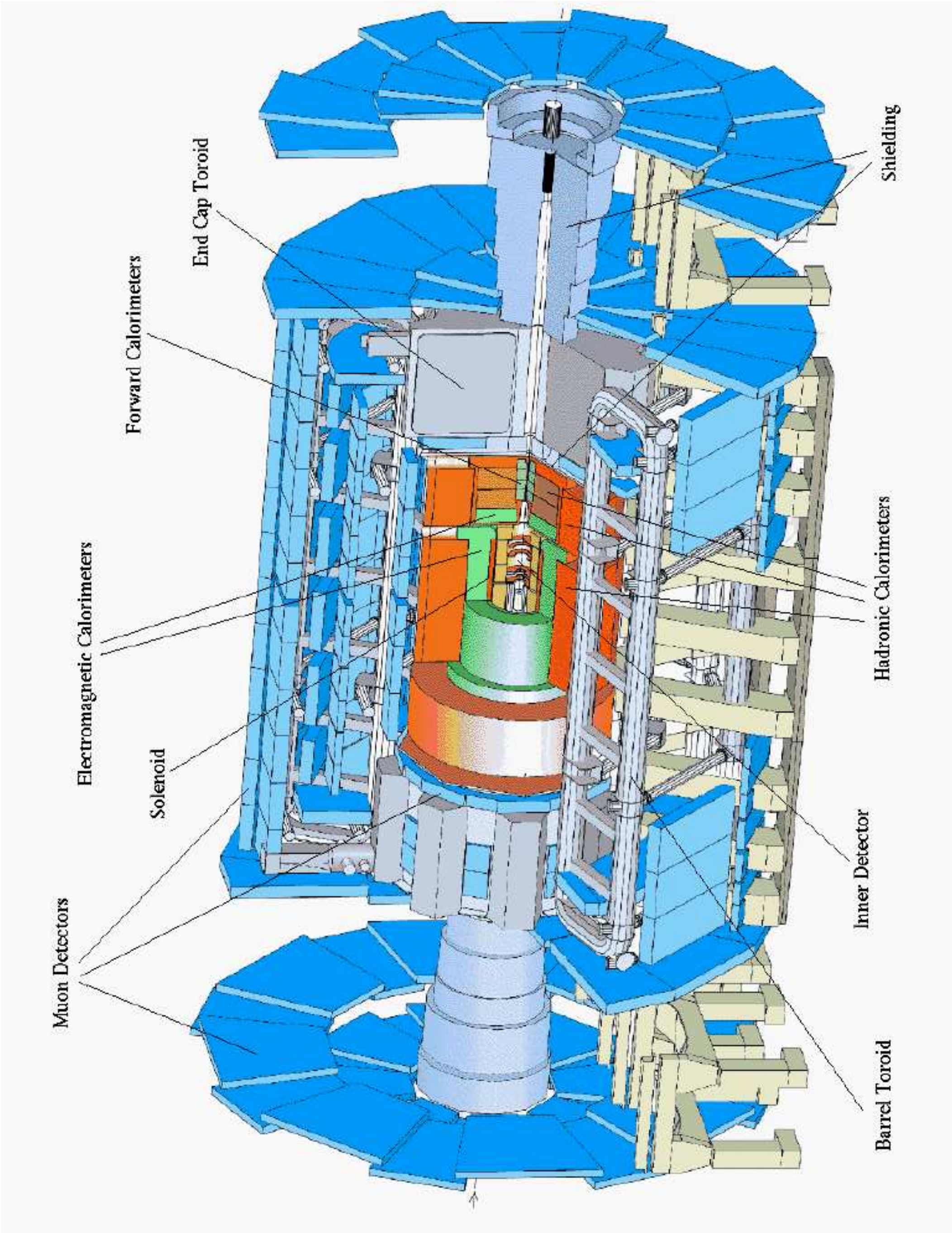

Figure 3.1: The overview picture of the ATLAS detector.

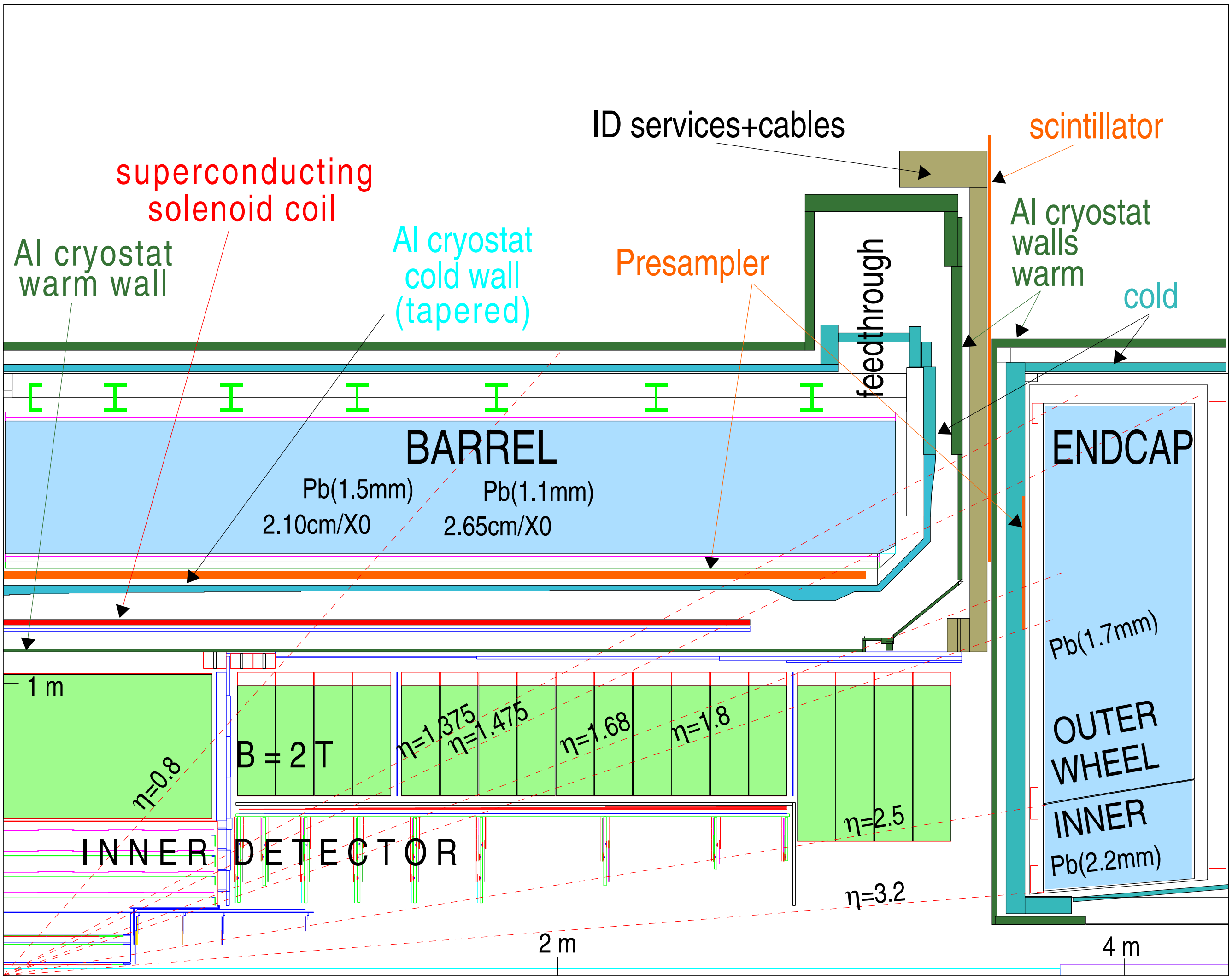

Figure 3.2: The ρz cross-section of the ATLAS detector.

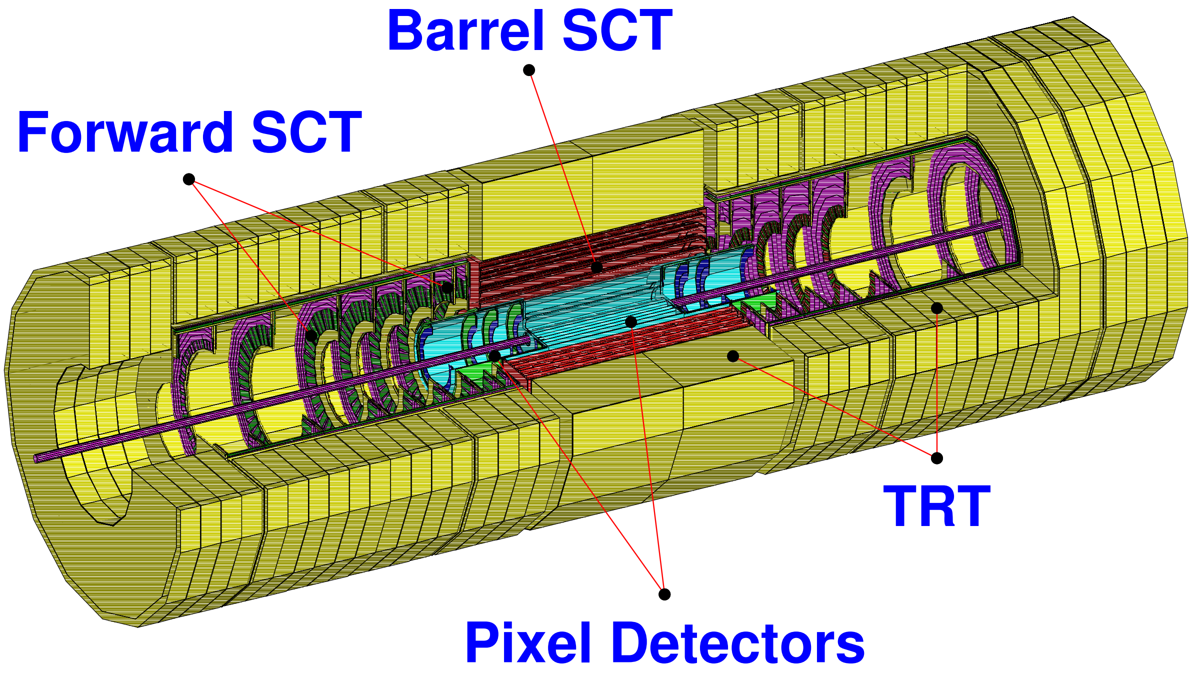

Figure 3.3: The ATLAS Inner Detector.

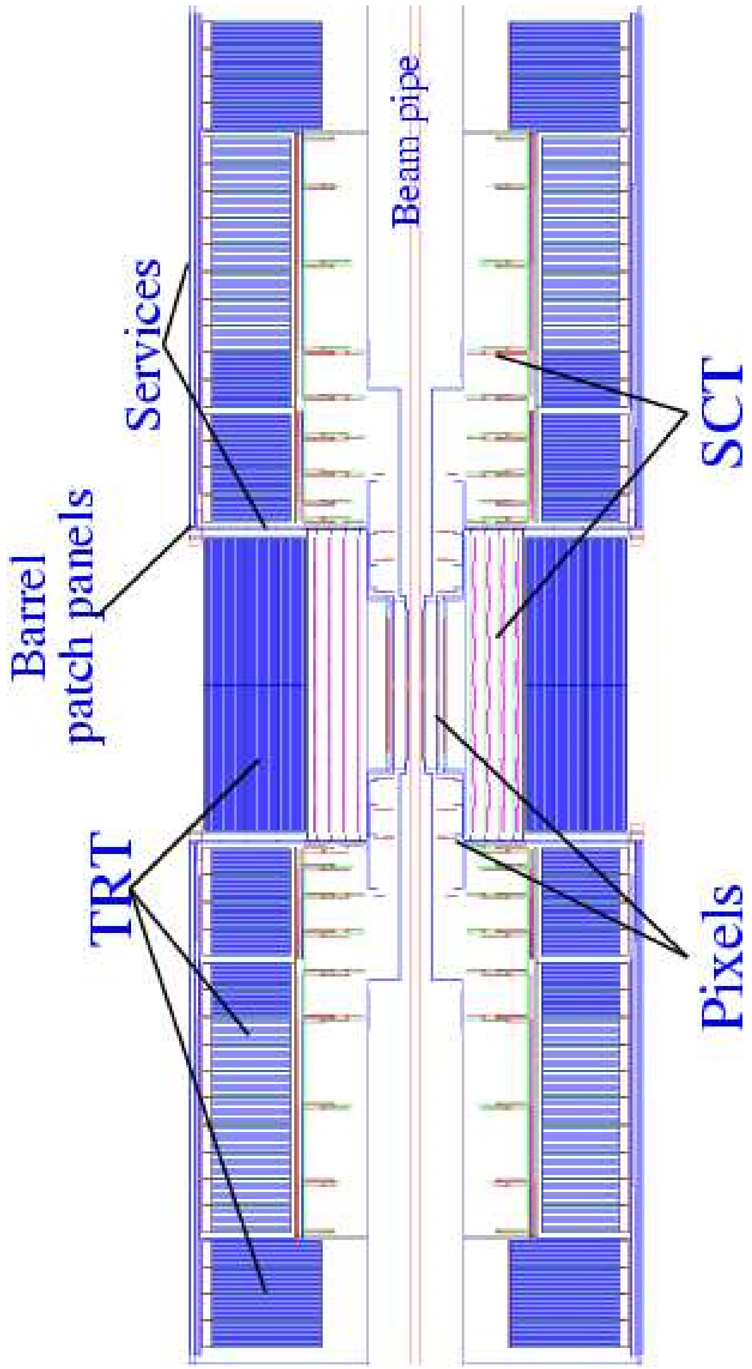

Figure 3.4: The ρz cross-section of the ATLAS Inner Detector.

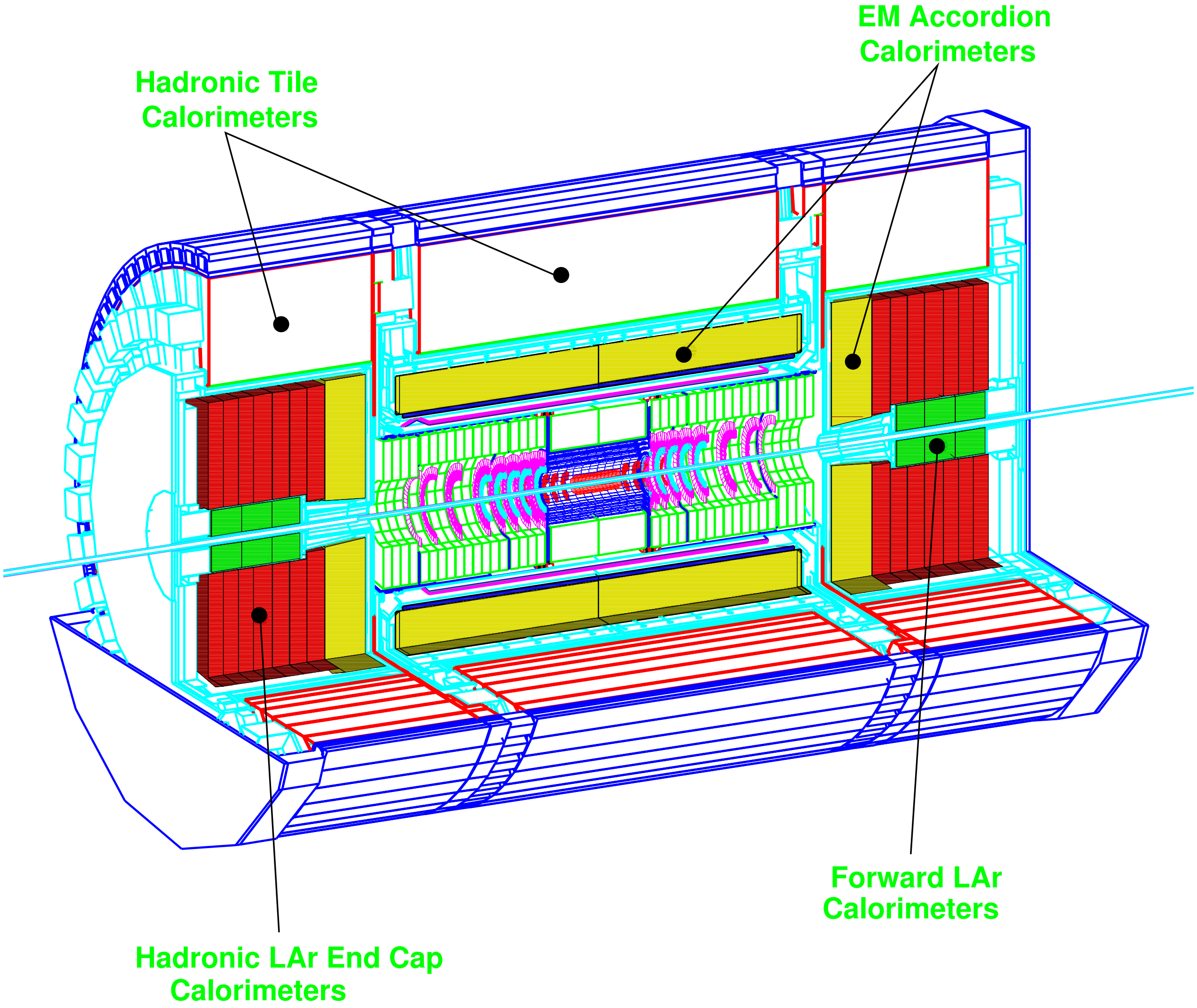

Figure 3.5: The ATLAS Calorimeters.

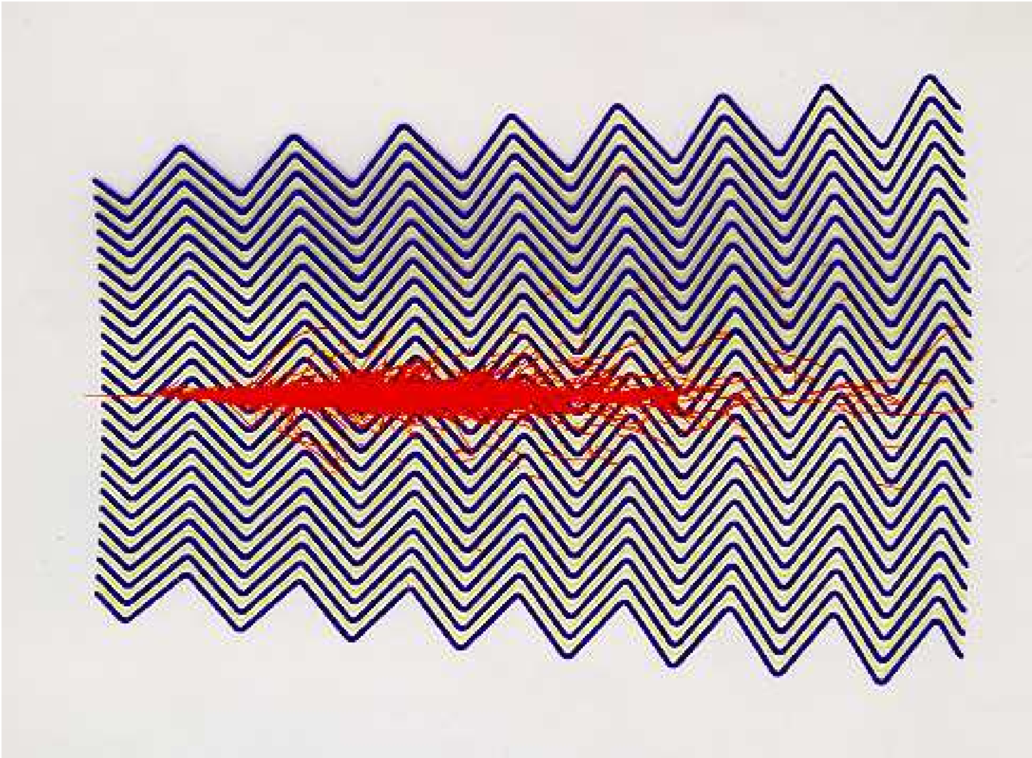

Figure 3.6: A shower in the LAr Calorimeter.

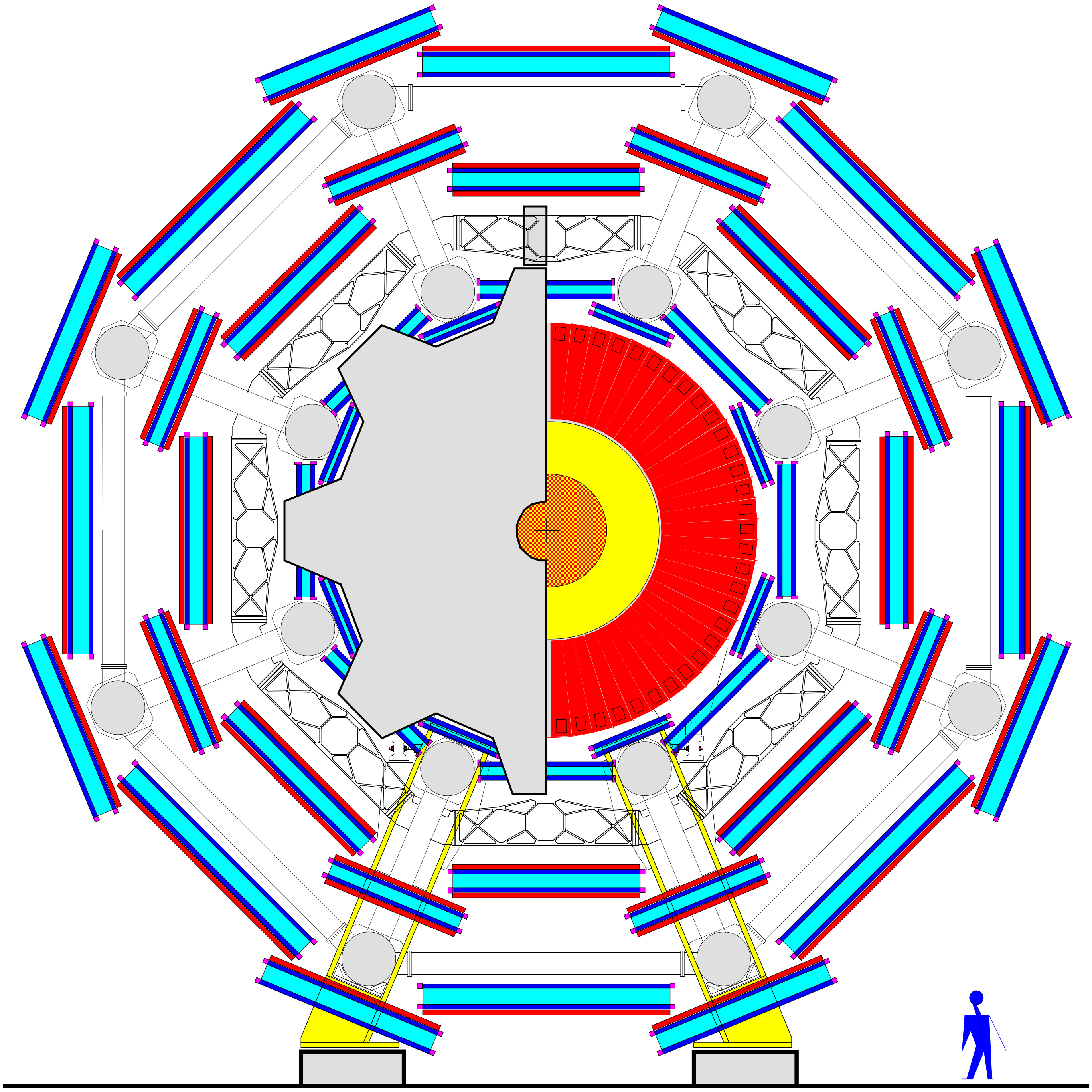

Figure 3.7: The ATLAS Muon Spectrometer.

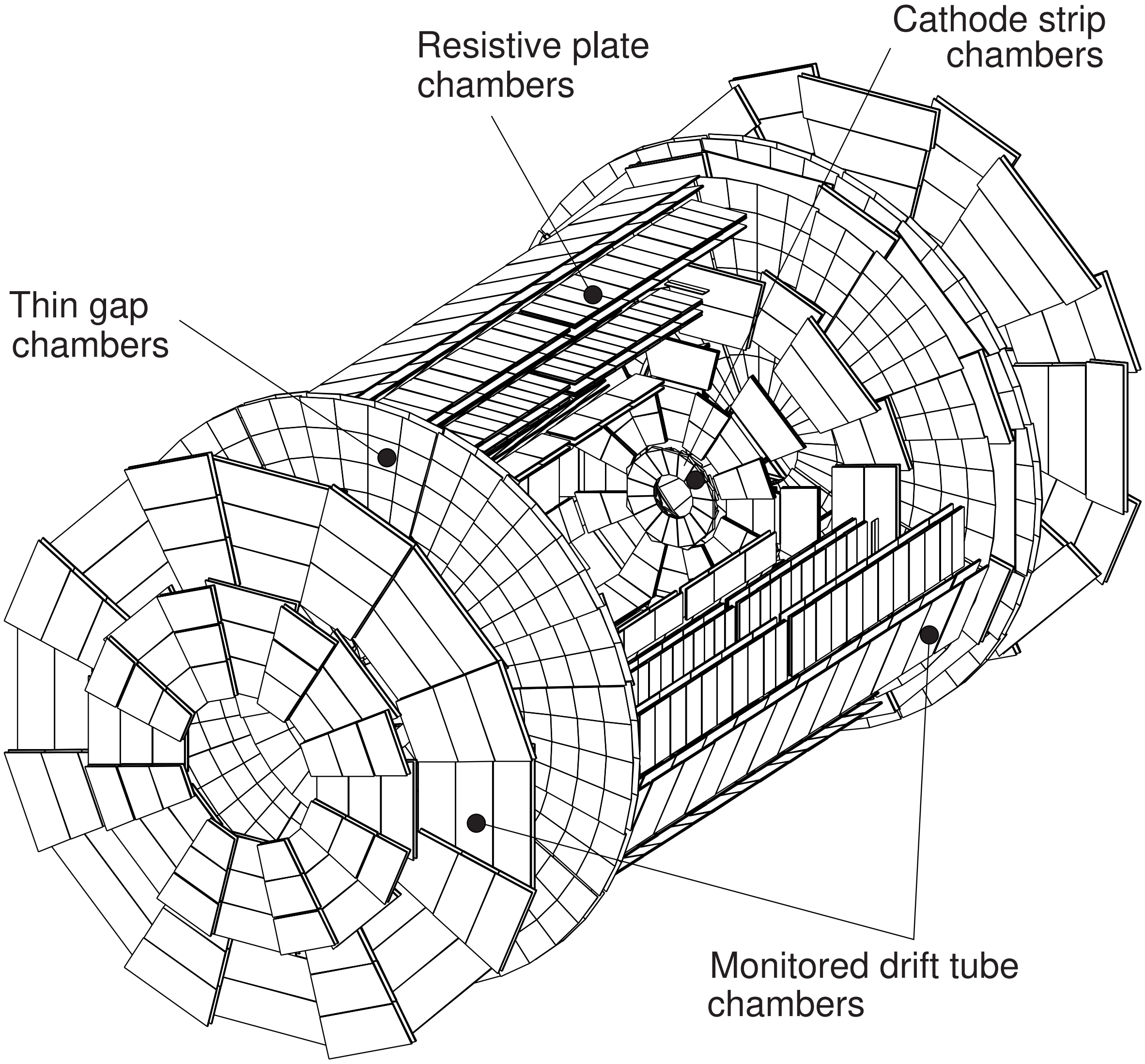

Figure 3.8: The ATLAS Muon Spectrometer.

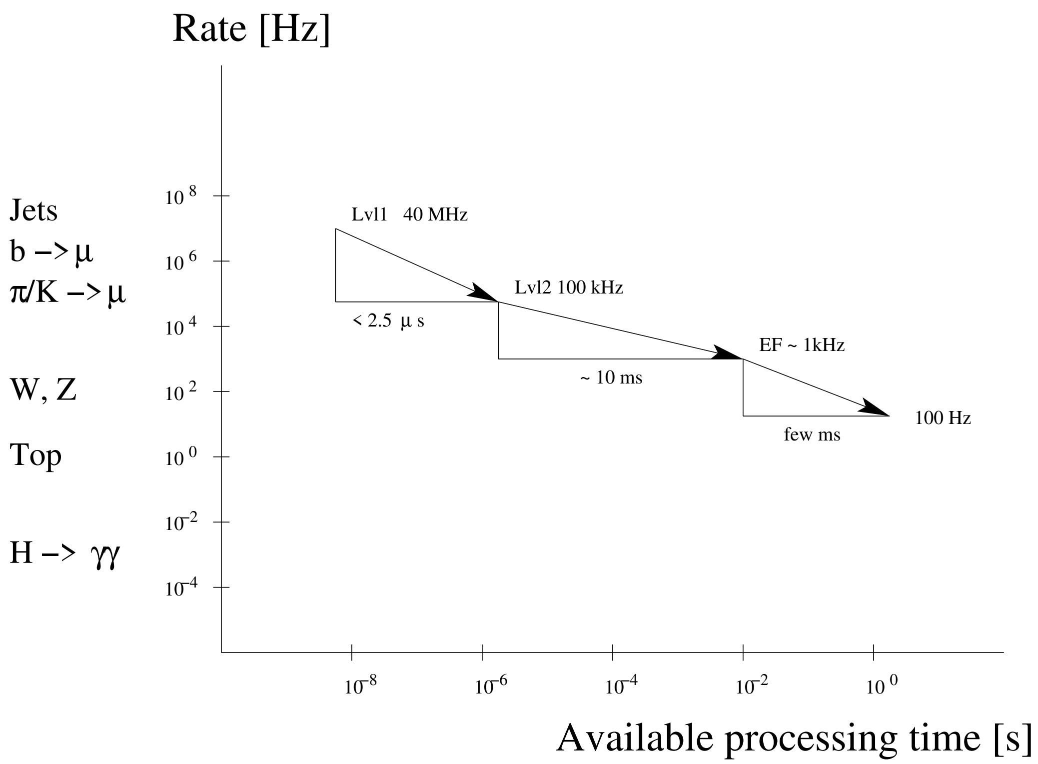

Figure 3.9: The three levels of the ATLAS trigger and their event rates and processing times.

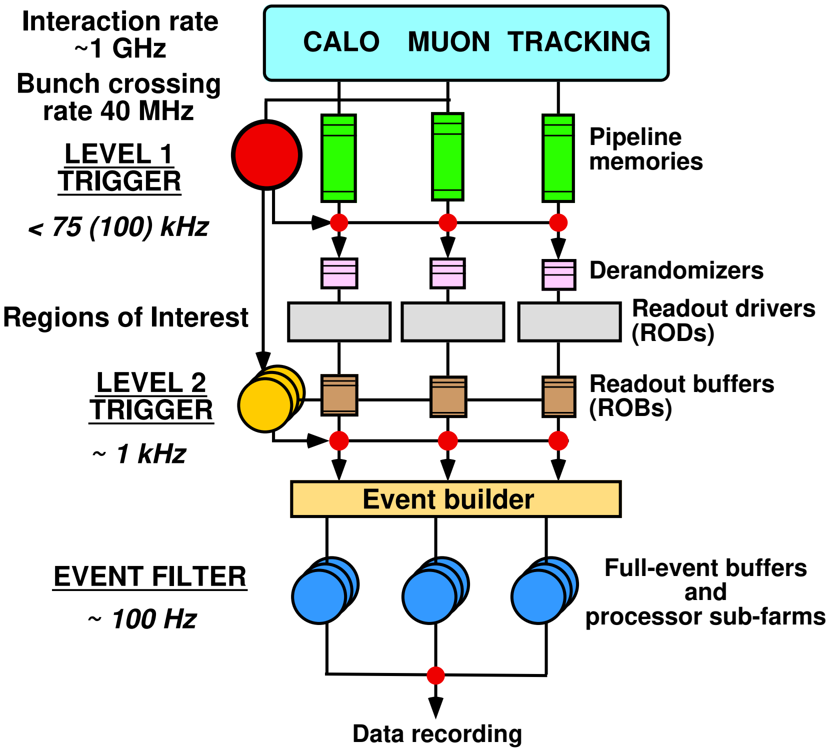

Figure 3.10: The ATLAS Trigger chain.

Chapter 4

Software

This Chapter deals with software issues in general. It describes relevant Programming Methodologies, Software Processes, Tools and Programming Languages of interest to the High Energy Physics. Most of the chapter is just compilation of various topics, some of which were, however, not available in such a distiled form to the HEP community until now. The most important (and to large extend original) contribution of the author is a set of general Software Design Rules for the Object Oriented Software Development (Section 4.1.2). Also guidelines for applying Extreme Programming Process (Section 4.2.7) are explained for the first time in the HEP community.

Software for LHC experiments will play more important role that software for previous generation experiments due to the amount and complexity of the data. Complex software in 3rd level trigger will make irreversible changes to the data. All these conditions put very strict requirements on the software quality. These requirements can be hardly satisfied by the software very often inherited from previous experiments without real understanding of its functionality.

Software development techniques relevant to HEP are shortly summarized in this chapter. Some of them are just of historical relevance, other are being newly exploited or can be used in near future. Only basic sketch of Software Development topics is given here, details are available from references.

4.1 Programming Methodology

Many paradigms of actual programming process have been developed and used. They are quite often coupled with (a set of) programming languages, sometimes they fit together with various Software Processes described in 4.2.

HEP is currently on the migration from the Procedural Programming (using mostly Fortran as the implementation language) into the Object Oriented Programming (implemented in C++ or Java). Even if HEP will not directly use other Programming Methodologies (Subject Oriented Programming, Aspect Oriented Programming, etc.), one should be aware about them and prepared to apply their techniques in the HEP environment, where appropriate.

4.1.1 Procedural Programming

For many years, one could only find procedural and imperative programs in HEP, using mostly various flavors of Fortran [43], sometimes (in online) also C [15]. Impressive amount of extremely high quality code has been created and widely used in HEP. CERNLIB [56] is considered as one of the most successful Fortran libraries.

It seems, that HEP has pushed Procedural Programming to its extreme (or even beyond it) and it has become obvious that it can hardly evolve further and in order to describe increasingly complex detectors and physics other paradigm has to be chosen.

The migration from Procedural Fortran code in HEP has been found very painful mostly due to that vast amount of highly sophisticated physics code (which, on the other hand, is often not enough documented and very difficult to understand). Equally important component of the difficulty in changing Programming Paradigm is apparent lack of HEP people with formal training in software technology and often lack of interest to acquire it.

4.1.2 Object Oriented Programming

During lifetime of a software project, only less then 30% of the effort goes into the system development, more than 70% is spent on maintenance. It has been observed, that "entropy of the computing system" grows with its lifetime till it reaches the asymptotic region (symbolically drawn on Fig. 4.1). 20% of all systems reach this region before their first version is delivered. Object Oriented technology promises to make the maintenance easier and to push the asymptotic limit further into the future (but it can't completely remove it).

The experience from the traditional HEP software indicates very clearly that current software methods don't scale and that different way of software paradigm has to be adopted. More declarative Object Oriented Technology, now already widely used in industry is the most logical candidate together with application of the Software Engineering Methods.

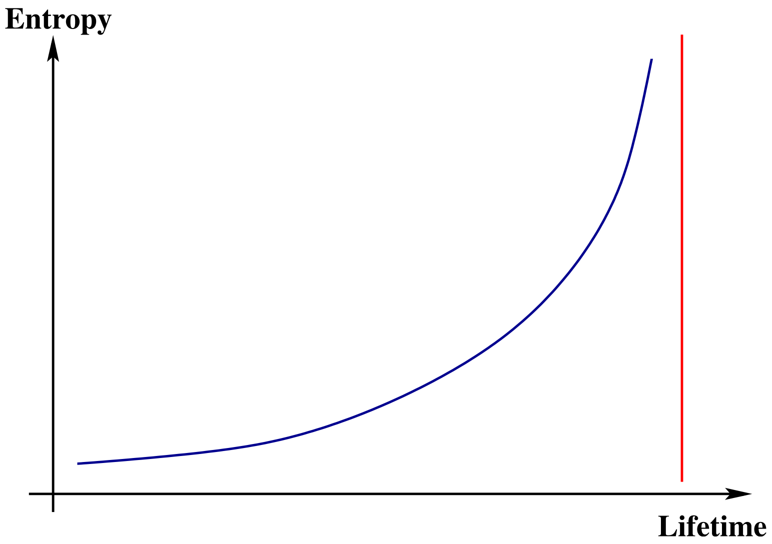

Figure 4.1: Entropy of a computing system as a function of its lifetime (Artistic' Drawing).

Object Oriented (OO) programming is relatively new approach how to write software in HEP. Generally speaking, OO program doesn't consist of functions, but of objects.

The basic concept of the Object Oriented Technology is to divide program into objects, which don't share any common data (objects are defined by their classes). This concept is called **Encapsulation** because it encapsulates data and the way, how they are manipulated inside the object. The data are even hidden from the outer world, so other objects can't mess them (**Data Hiding**). The rest of the program doesn't have to know, how is the particular object implemented and is therefore not influenced by the change of its internal representation or implemented algorithm. It only interacts with well-defined object interface. An Object is schematicaly described on Fig. 4.2.

This is different approach to the way procedural languages work - their programs usually consist of routines which interchanges openly its data and very often even work on common data. This makes the data very vulnerable as they can be changed from everywhere and even small change in some routine or data definition influences in principle everyone.

While Object Oriented software has been shown to be more maintainable, changeable and readable, it's very often found to be more difficult to write. This is why it usually takes longer to deliver the first version of the OO system, but evolution to the higher versions is simpler.

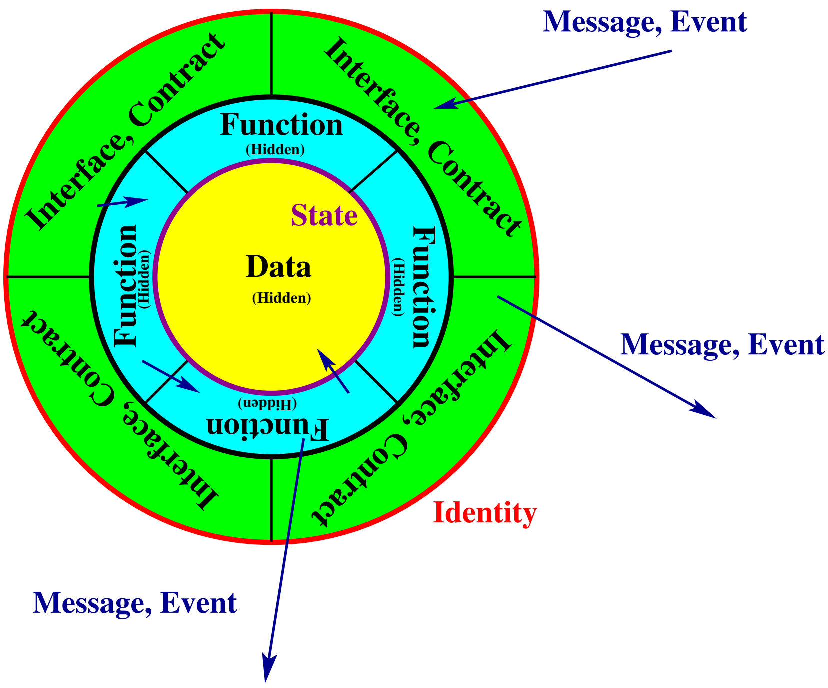

Figure 4.2: **Object** = **Data** + **Contract** (Interface, Service). It has its **Identity** (seen from outside) and **State**. *Object is a data structure with attitude [John Deacon].*

Object Orientation fits better in *Modeling the World* (which is Object Oriented) but is often harder to invent for Human Being (which seems to be procedural). The OO software development is easier when following known OO Rules [Robert C. Martin, Elliott Coates, John Deacon, Edward Berard, ...]:

Open-Closed Principle Software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification. New functionality should be added by adding new code not by editing old code.

Common Closure Principle The classes in a package should be closed together against the same kind of changes. A change that affects a package affects all the classes in that package.

Common Reuse Principle The classes in a package are reused together. If one reuses one of the classes in a package, she reuses them all.

Interface Segregation Principle Clients should not be forced to depend upon interfaces that they do not use.

Liskov Substitution Principle Functions that use references (or pointers) to base classes must be able to use objects of derived classes without knowing it.

Dependency Inversion Principle High level modules should not depend upon low level modules. Both should depend upon abstractions. Abstractions should not depend upon details. Details should depend upon abstractions.

Acyclic Dependencies Principle The dependency structure between packages must be a Directed Acyclic Graph. That is, there must be no oriented cycles in the dependency structure.

Law of Demeter Each unit should have only limited knowledge about other units: only units closely related to the current unit should be known. Each unit should only talk to friends. *Don't talk to strangers.* A method should have limited knowledge of an object model. Inside a method, it is only permitted to access or send messages to the following objects:

- Arguments associated with the method (incl. `self`).
- Instance variables defined in the class containing the method being executed.
- Global variables.
- Temporary variables created inside the method.

Violation is characterized by requesting an Object and then calling its method.

Functionality Rule The classes should be partitioned according to their use case functionality into packages.

Boundary - Controller - Entity Rule The classes in a package should be divided into Boundary, Controller and Entity classes.

Specialization and Generalization Rule Redundant behavior and responsibility should be taken out of the classes where it is found and placed in a single class.

Coherent Responsibility and Interface Rule The interface to a class should concentrate on providing a highly related set of operations.

Hierarchical, Modular and Layered Rule Large system should be built in layers. Within layers, classes should be organized hierarchically, with top level classes providing an interface to the layer, and the higher levels making calls to lower levels to fulfill detailed implementation. Within a layer, code should be organized in loosely coupled and highly self-related classes.

Design by Contract Each major calls should have pre and post-conditions.

Encapsulation Rule Interface should be separated from implementation.

Localized Dependency Rule Each entity should be defined in one place.

KISS Rule *Keep it simple, stupid.* Each functionality should be implemented in the most simple possible way.

Tell, don't Ask Procedural code gets information then makes decisions. Object-oriented code tells objects to do things. *Violation is characterized by exposing Iterators instead of accepting orders.*

Use Cases Role Use Cases are used to describe a public interface of the system, not as a basis for the description of its internal structure. *Violation is characterized by massive use of "Control Objects" or "Managers" and parallel development of the Designs with the same functionality, but different implementations, which are difficult to merge later.*

Some rules are applicable also to other programming techniques and there is certain overlap of the rules into the Software Process.

OO Pioneers in HEP have tried Objective C [16] and Eiffel (see 4.4.1) as the OO Programming Languages, but currently the most popular is C++ (see 4.4.2). Java (see A) is steeply gaining its popularity in HEP and is expected to play very important role in near future.

4.1.3 Evolution of the Object Oriented Programming

Object Oriented programming, while being in most cases big step forward in the software architecture and design, has obviously its limitations. Some of them are mentioned in this section, together with possible solutions.

Subject Oriented Programming [17]

For an OO program, a class is often too fine level of granularity to capture a mission (or one of the missions) of a system. To solve the problem, classes can be organized into collections - *subjects*. Organizing classes into subjects allows to solve several problems common in the HEP software development, like parallel development in a distributed environment or rapidly changing software.

SOP is not available from any mainstream programming languages, it is only supported via language extensions to C++ or Java.

Aspect Oriented Programming [18]

AOP tries to solve the problem of concerns which cross Domain boundaries (**Cross-cutting Concerns**). Example of such concerns are error-checking, resource sharing or Design Patterns. Cross-cutting concerns are very frequent in the HEP code, where per-detector domain decomposition is often orthogonal to the domain decomposition of the framework itself. **Aspects** are units of software modularity that cleanly encapsulate cross-cutting concerns.

AOP isn't currently supported by any mainstream programming language, the only support is provided via Java preprocessing (AspectJ [19]).

4.1.4 Design Patterns

Design Patterns [20] is a successful technique for managing complexity of the OO programs by factoring out the core of solution to frequent problems. Standard Patterns are documented and even some CASE Tools are able to create Pattern skeletons. Support for some patterns is directly available from some languages (like *Iterator* or *Observer* in Java) or standard libraries (like in C++ STL [21]). Using Design Patterns also simplifies low-level documentation as it is not necessary to document patterns itself, it's enough just to name it.

Among frequently used Design Patterns in HEP are *Abstract Factory*, *Adapter*, *Iterator*, *Observer*, *Strategy* and *Visitor* Pattern.

4.2 Software Process

Many methods have been invented which may help to manage Object Oriented programming:

4.2.1 Entity Relationship Model - ER

ER [22] describes only data, their relations and how to process them. It has been used by the ALEPH and ZEUS experiment and is still used in analysing data for Relational Databases.

4.2.2 Class Relation Methodology - CR

CR [23] is based on the ER model. It has the notion of schema to group classes and to allow classes to be hidden inside a schema. It has also dynamic and functional model inside.

4.2.3 Object Oriented Modeling Technique - OMT

OMT [24] consists from the *Object Model*, *Dynamic Model* and *Functional Model*. The *object model* describes the static part of the system, its parts and relationships. It is similar to the ER. The *dynamic model* describes the behavioral aspects of the objects by nested Finite State machines showing the transitions between states. The *functional model* is just traditional data flow diagram. OMT is strong in describing high-level Design, but weaker in Analysis and detailed Design.

4.2.4 Booch Method

Booch Method [25] is similar to OMT. It differs in diagram syntax and process definition. It is stronger than OMT in low level Design and Analysis, weaker in high level Design.

4.2.5 Rational Unified Process - RUP

Process is about managing fear. We put processes in place because we are afraid. If our fear is large, we put big processes in place. [Robert C. Martin]

RUP [26] is nowadays the leading method, supported by all major CASE tools. It puts together better features of OMT, Booch and other methods.

RUP is an *iterative* process, which means that the understanding of the problem and solution are improved over successive incremental development cycles. RUP is *architecture-centric*, it focuses on the early development and baseline of the software architecture. It is *use-case driven*, it is based on the understanding how a system will be used.

RUP consists of the following four phases (shown in Fig. 4.3):

Inception Establish the business case for the project.

Elaboration Establish project plan and a sound architecture.

Construction Grow the system.

Transition Supply the system to its end users.

Within each phase, there is a number of iterations, each representing a complete Development Cycle (shown in Fig. 4.4):

- **Planning**
- **Analysis**
- **Architecture**
- **Design**
- **Implementation**
- **Integration**
- **Testing/Assessment**

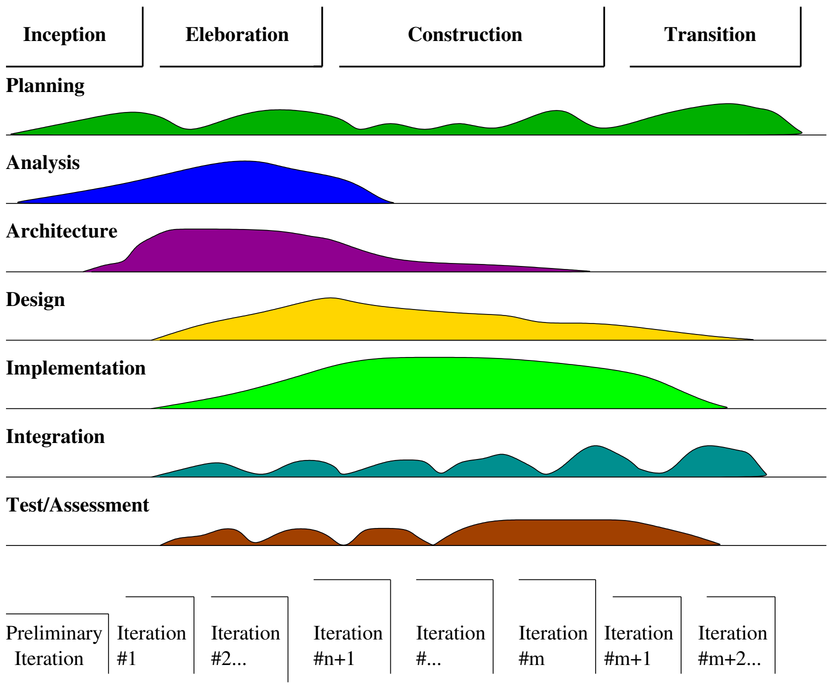

Figure 4.3: The Software Development Phases.

RUP uses graphical **Unified Modeling Language - UML** to visualize all artifacts of a software process. While there are other processes in use today (see later), almost everyone is using UML.

The UML includes nine kinds of diagrams:

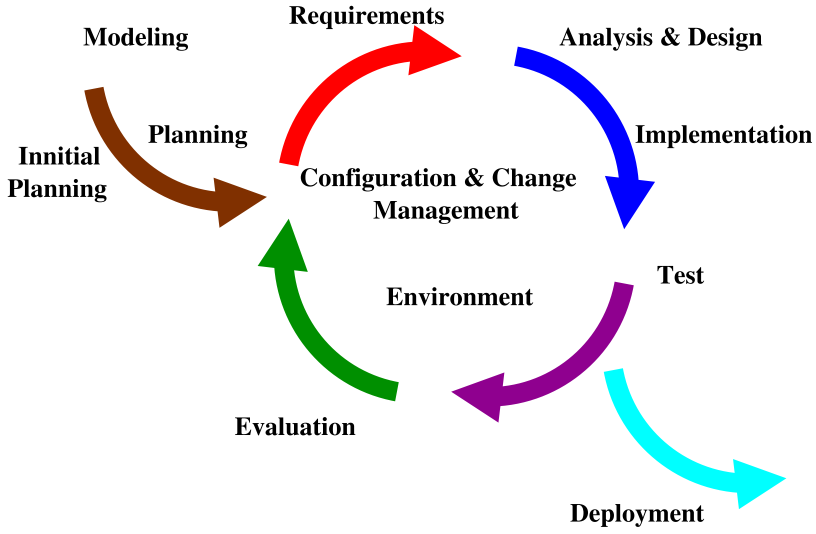

Figure 4.4: The Software Development Life Cycle.

Class diagram A structural diagram that shows a set of classes, interfaces, collaborations, and their relationships.

Object diagram A structural diagram that shows a set of objects and their relationships.

Use Case diagram A behavioral diagram that shows a set of use cases and actors and their relationships.

Sequence diagram A behavioral diagram that shows an interaction, emphasizing the time ordering of messages.

Collaboration diagram A behavioral diagram that shows an interaction, emphasizing the structural organization of the objects that send and receive messages.

Statechart diagram A behavioral diagram that shows a state machine, emphasizing the event-ordered behavior of an object.

Activity diagram A behavioral diagram that shows a state machine, emphasizing the flow from activity to activity.

Component diagram A structural diagram that shows a set of components and their relationships.

Deployment diagram A structural diagram that shows a set of nodes and their relationships.

Examples of some of those diagrams are shown later.

4.2.6 Lightweight Methodologies

There is one sure way to fail to produce a reusable framework; and that's to design in up front. This strategy has been repeated manymany times, and the results are frequently the same: the reusable framework isn't very reusable. [Robert C. Martin]

As a reaction to the complexity and formality of traditional heavy-weighted (or *monumental*) processes (RUP), several lightweight processes have been created. The offered solution is not to abandon Software Process, but to make it useful. They don't advocate return to the previous *code and fix* style of software development. Light methods manifest generally two major differences to the heavy ones:

- They are **adaptive rather than predictive**. Heavy methods tend to plan everything. However, in a world of rapidly changing requirements and conditions (the world of the software development), long term planning is not really possible. Heavy methods are not prepared for a change - they resist it. Light methods, on the other hand, are based on the notion of change and adaption.

- They are **people-oriented rather than process-oriented**. Heavy methods rely of formalities, paperwork. Light methods are dealing with humans and emphasize that.

4.2.7 Extreme Programing - XP

Extreme Programming [27] is quite new lightweight Software Process. While it is not (yet) formally used in HEP, it seems to be the Extreme Programming which fits naturally into the HEP environment and many successful HEP programmers are spontaneously using it (or at least some of its features).

The Extreme Programming Process can be roughly described as a several-stage process:

1. Getting **User Stories**.

2. Creating **System Metaphor**, which involves **Understanding** of the User Stories, problem Analysis and a bit of Design. System Metaphor is a seed for an Architecture.

3. Creating of the **Test Cases** which describe the the Understanding. One has certainly a rough view of a Design in mind, but the important is that the Test Case creation comes before the actual Development.

4. **Coding** to satisfy Test Cases (and nothing more) contains Designing as a part of its activity.

5. **Refactoring** leads to simpler and cleaner code by removing redundant, not understandable or useless code.

There are several key rules to be followed in the Extreme Programming Process (they are sometimes called *enablers*):

Simplicity The code to perform the task should be the most simple, clean and flexible. The code should be the simplest possible to satisfy the Test Cases.

Communication The most important part of this rule is programming in (unstable) pairs of programmers working on the same piece of code.

Reassesement It should happen in the beginning of each iteration.

You Aren't Going to Need It - YAGNI There should be no long speculation about the future code. They would be mostly wrong.

Group ownership of the code Code doesn't belong to individuals (or to the pairs, which are after all unstable) but to the whole team (group), everybody is concerned.

No separation of roles There are no Architects, Analysts, Designers or Coders. Everyone plays all roles for his piece of project.

Readability The code should be readable. If it isn't, the solution is not to write extensive documentation, but to improve the code. *Clever Code* is a term of abuse.

Design by Testing Test case should be done first. The code should then be designed just to satisfy the test case.

Load Factor The productivity of development should be measured and ration between the estimations for the duration of the task and the reality should be applied in future cycles.

Short Cycles One cycle should not take longer than three weeks.

Worst Things First The most difficult part should be done in the beginning of the project.

Continuous Integration The code should always pass also integration tests.

Refactoring, Don't Repeat Yourself - DRY Every task should be implemented once and only once. The should be no duplication.

4.2.8 Feature Driven Development - FDD

The ultimate judgment of progress is this: measurable results in reasonable time. [Robert Anthony]

Feature Driven Development [28] is a model-driven short-iteration process. It begins with establishing an overall model shape. Then it continues with a series of two-weak *design by feature, build by feature* iterations.

A useful small *feature* is a little tiny building block for planning, reporting, and tracking. It's understandable, it's measurable, it's do-able (with several other features) within a two-week increment.

FDD consists of five processes:

- **Develop an overall model** using initial requirements/features, snap together with components, focusing on shape.
- **Build a** detailed, prioritized **feature list**.
- **Plan by feature.**
- **Design by feature** using components, focusing on sequences.
- **Build by feature.**

FDD is supported by the Together CASE Tool (see 4.3.3).

4.2.9 Bazaar

Bazaar [29] style of development could be considered as a Software Process as well. Bazaar style has been described as the process, which is used to develop Open Source [30] Software (GNU [31], Linux [32], etc.).

The major characteristics of the Bazaar style of development (in contrast to traditional "Cathedral style" is the existence of a very small core development team, but a very large community of User-Developers-Debuggers. Those are the people who really do most of the debugging and even development (after the initial kick-off version).

4.2.10 Capability Maturity Model - CMM

Carnegie Melon Software Engineering Institute (SEI) [34] Capability Maturity Model [33] defines five levels of software process maturity. CMM is, however, often felt to be too formal and not targeted to real problems (*The processes in use today focus on paper rather than people. Paper can't think, paper can't solve problems. Paper can't adapt to change. [Robert C. Martin]*). CMM is often seen as a typical case of a heavy weight style of software development (and SEI really advocates such application of it). The notion of software process improvement, however, is absolutely valid and CMM levels can be formulated in a way acceptable also for lightweight processes:

1. **Initial Level:** The software process is characterized as ad hoc, and occasionally even chaotic. Few processes are defined, and success depends on individual effort and heroics. The project is planed. *Violation is characterized by: "We prefer nasty surprises halfway through."*

2. **Repeatable Level:** Basic project management processes are established to track cost, schedule, and functionality. The necessary process discipline is in place to repeat earlier successes on projects with similar applications. Relevant information is recorded persistently. Ad-hoc process eventually becomes a set of entrenched practices that are repeatable. *Violation is characterized by: "We prefer the unreliability of human memory."*

3. **Defined Level:** The software process for both management and engineering activities is documented, standardized, and integrated into a standard software process for the organization. All projects use an approved, tailored version of the organization's standard software process for developing and maintaining software. Stakeholders are informed about changes. Repeatable practices are written down so that others can learn them. *Violation is characterized by: "We can't be bothered with other people's problems."*

4. **Managed Level:** Detailed measures of the software process and product quality are collected. Both the software process and products are quantitatively understood and controlled. The process is monitored. The organization begins to measure the way the process works. *Violation is characterized by: "We see no need to learn from past experience."*

5. **Optimizing Level:** Continuous process improvement is enabled by quantitative feedback from the process and from piloting innovative

ideas and technologies. The process should is being improved. The organization applies process measurements to improving the process. *Violation is characterized by: "We are already perfect."*

4.3 CASE Tools

The CASE (Computer Aided Software Engineering) tools provide a graphical user interface to support some of the above mentioned Methods. They generally support object model describing the classes, their relations and status. They can usually be used for the assignment of attributes and methods to the classes. Often, they can generate code from diagrams or reverse-engineer code into diagrams.

4.3.1 Rational Rose

Rational Rose [35] is one of the leading CASE tools in the market. It is very user friendly, it isn't very configurable. Its very high price makes it very questionable for HEP. Rational Rose supported originally Booch method, now it uses RUP with UML. Rational Rose is used for example by the Geant4 [57] project.

4.3.2 Software Through Pictures - StP

Software Through Pictures [37] was a tool previously used in ATLAS. It supports all Booch, OMT and UML methods. It is not very user friendly, but allows customization, which could be used to generate code and Design reports suitable for ATLAS collaboration.

4.3.3 Together

Together [38] is a lightweight CASE tool for development of C++ and Java programs. Its simplified ("whiteboard") version is available for free. Together is completely portable (as it is written in Java) and has a unique feature that it works directly with the code so no code-generating or reverse-engineering phases are necessary. Together contains comprehensive code-checking and metrics functionality. It is now the recommended tool in ATLAS. Most of the UML diagrams shown below have been created with Together.

4.3.4 ArgoUML

ArgoUML [39] is easy to use OpenSource java-based CASE Tool for Java. It offers some unique features of *cognitive support* to facilitate software development (using design comments etc.).

4.4 Languages

The question of languages is in principle not as important in the OO world as it was in the world of procedural languages. Especially with various CASE tools, detailed syntax of the program can be let on the computer and programmer (designer) may fully concentrate himself to the higher levels of the software development.

Only languages (of various types) relevant to the HEP software environment are mentioned here.

4.4.1 Eiffel

Eiffel [40] was designed to follow pure OO principles. It allows a close link between design and implementation. Its main features include simple syntax, parametrised classes, assertions, exception handling, properties (which can be either parameter-less functions or data) and automatic garbage collection. *Design by Contract* implemented by Preconditions and Postconditions, is integral part of the language. They implement assertions upon entering and leaving a method. If a precondition fails upon entering a method, or if a postcondition fails when leaving, an exception is raised. Eiffel is still the language which implements the OO concepts most cleanly.

4.4.2 C++

C++ [41] is one of supersets of C (the other one, Objective C [16], has practically disappeared). C++ provides (among other features) complete Object Oriented environment. It has on the other hand also very complicated syntax and is far from canonical so it does not enforce good programing style. In C++ “everything is allowed” which leads often to unreadable and unmanageable code. C++ doesn’t have context-free grammar (similarly to Fortran) which creates many problems in building interactive C++ environment. C++ is, however, by far the most commonly used language in the software industry so many people know it and there are many class libraries and tools developed for C++.

4.4.3 Smalltalk

Smalltalk [42] is an interpreted language. It is both a language and a software development environment. It is pure OO language - everything is viewed as an object (even numbers).

4.4.4 Fortran 90

Fortran 90 [43] is not an Object Oriented language as it allows to use only datastructures as “objects”. It is aimed towards mathematically intensive computing (like massive vector processing). The structure of HEP software tasks makes Fortran unsuitable for major offline projects. On the other hand, having in mind long tradition of Fortran programming in HEP and all existing legacy code, it’s probably necessary to allow Fortran code to interoperate with major packages. However the core software should be designed in the Object Oriented way.

4.4.5 Perl, Python, Tcl/Tk

Perl [44], Python [45] and Tcl/Tk [46] are *scripting languages* suitable for writing frontend for other packages. Python is fully Object Oriented, Perl and Tcl/Tk have standard Object Oriented extensions. Interaction between scripting and compiled programming languages is possible.

4.4.6 Extensible Markup language - XML

XML [47] is the most widely used *descriptive language* suitable for describing data structures (in Ascii format). In HEP, XML is used to store detector geometry (as in AGDD [64]), Events (as in WIRED [83]) histograms (as in AIDA [61]) and program configuration (as in JAS [86]).

4.4.7 Virtual Reality Modeling Language - VRML

VRML [48] is high level *descriptive language* for 3-dimensional objects. It is well Object Oriented and allows interoperation with HTML language. Existence of good browsers and their possibility of interacting with other packages makes VRML good candidate for the Event Display language sitting on top of the offline packages. VRML 2 (“Moving Worlds”) offers the possibility of creating interacting behaving objects (VRML 3D object can be connected to Java object). The syntax of the VRML has been rewritten using XML language making this way all XML tools available for VRML (see X3D [49]).

4.4.8 Java

Java [50] is a relatively new programming OO language. Java uses very clean OO concept, most confusing features of C++ are absent in Java (multiple inheritance, pointer arithmetics,...). Java seems to be a successor to C++ in HEP. More details about java are available from Appendix A - **On the importance of Java to ATLAS**.

Chapter 5

ATLAS Software

This Chapter concerns the ATLAS software. After a short description of the ATLAS software status (with a short analysis of the situation), three major software topics are covered. Section 5.3.2 describes original contribution to the ATLAS Software Process - The Handling guidelines for the External Software. The main original software contribution is described in the Section 5.4 - it covers the development of the ATLAS Graphics Systems. All relevant phases of the development are described: From the accumulated User Requirements (summarized in the Appendix B), through the top level Graphics Architecture to the Design of the key graphics component - the Graphics Control. The author of the thesis has coordinated the development of the Graphics Framework, has defined and developed its Architecture and has developed some of its major components (the most important is the overall Graphics Control Structure). All graphical components (Scenes) are described and the emphasis is laid on the Scene developed by the author of the thesis - the 3D Event Display GraXML. Also author's contributions beyond ATLAS, into FreeHEP library and HEP Studio, are mentioned. This Graphics Framework currently constitutes the baseline of the ATLAS Graphics environment and is integrated into the overall ATLAS Framework - Athena. The ATLAS Graphics system has been presented in several international conferences. Section 5.5 shortly mentions the original algorithm for the fast Pattern Recognition developed by the author, the implementation of it, however, hasn't gone beyond a prototype level.

5.1 Traditional System

Current (old) offline software in HEP is mostly written in Fortran 77. As it has been often created without any design by simply adding additional features to already existent functional programs, it is in a very unpredictable state and no-one really knows what may happen when such software runs.

5.1.1 Software Packages

The beginning of the ATLAS offline software dates back about 10 years, when the project EAGLE started in the UA2 Collaboration. The basic structure of the software is thus influenced very heavily by the UA2 offline software.

All past ATLAS offline software is based on Fortran 77. It uses the traditional HEP software packages as Geant3 [58] for detector simulation, Pythia-Jetset [59] for event generation, Zebra [60] for access to data and CERNLIB [56] for access to basic library functions.

Main Fortran packages in use are:

SLUG [65] provides higher level functions to access and manipulate Geant3 datastructures.

DICE [66] serves the detector specific routines (geometry, digitisation,...) used by Geant3.

ATRECON [67] does all the reconstruction.

Age [68] is a language developed as a geometry specification language from which the Geant3 initialization is done. It has grown into un-predicted size and has polluted almost everything else. Age design has been heavily influenced by the **MORTRAN [51]** a language, which is language to be pre-processed to get Fortran code. This makes any code maintenance and debugging very painful.

ATLSIM [69] is a kind of interactive version of ATRECON.

ATRIG [70] contains the Trigger-specific code, with a functionality sometimes overlapping with other packages.

5.1.2 Code Organization

All packages were traditionally maintained in the form of CMZ [62] files in several versions. Also libraries and executables were created for each version. The most current versions were named `old`, `pro` as production, `new` and `dev` as development. The whole hierarchy was distributed via AFS [52] filesystem which enables access to it from the whole Internet.

5.1.3 Software Process

Strictly speaking, no Software Process has been defined for the old software. The reality could be probably best described as *chaotic best-effort based distributed software process*. Contributions to the common code base were done without any coordination only dictated by the short-term goals. The code, however, stayed forever and was constantly further used and changed.

5.1.4 Problems

The structure of the old ATLAS software was extremely difficult to maintain, change and very often even use. There are several sources of these problems (common in the whole HEP, not only in the ATLAS Collaboration):

Used Software Technology The technology used for the software development in the ATLAS Collaborations was very inadequate to the task. Sequential languages (as Fortran) are too heavily interconnected (for example via `COMMON`blocks or `EQUIVALENCE`s), that it's almost impossible to change the code consisting from many thousands lines of code without influencing other parts of the program. Also lack of data-structures in Fortran makes software developers to use very low level packages like ZEBRA [60] to maintain the the data. And because these packages are not a'priory known to the Fortran compiler, it cannot produce very efficient executable.

Management Attitude The HEP Managements paid only little attention to the software development keeping the old view of software as something not very important and not very difficult (" everybody can write any program and everybody does it - even without any training").

The importance of the software is now in reality much bigger than in past experiments. This is caused by the fact (among others), that software in LHC will be part of the trigger, that the LHC experiments will last longer than any software technology and effective lifetime of physicists and in the end by the amount and speed of date which will have to be processed (1 PB of date every year with the speed of 100 MB every second). The LHC experiments will be bigger and more complex than current HEP experiments, in detector size and complexity, number of physicists, event size, bunch crossing rate and almost certainly experiment lifetime. It is therefore crucial that all elements of an experiment should be designed, built and maintained to the highest engineering standards. Correctness of software has not been a major issue until now. It must be taken very seriously at LHC because complex software will be part of each LHC experiment's trigger.

5.2 Object Oriented System

While ATLAS has build already several prototypes of its Object Oriented software system (Arve [71], Paso [72],...), only the current implementation - Athena [76] - is described here.

5.2.1 Architecture

The Architecture of the ATLAS software Framework has been created by the *Architecture Task Force - ATF [73]* team. The team has decided to adopt the Architecture developed by the *LHCb [7]* experiment - *Gaudi [63]*. The work on the Athena is coordinated by the *ATLAS Architecture Team* lead by the *Chief Architect*. ATLAS has extended the original framework in several directions, some extensions are fed back to the Gaudi Framework. Common Gaudi core for several experiments (not just ATLAS and LHCb) is being created.

5.2.2 Software Packages

ATLAS OO software is organized into a matrix of packages according to the subsystem (Inner Detector, LAr Calorimeter, ...) and according to the functionality (Simulation, Reconstruction, ...).

5.2.3 Code Organization

All ATLAS software is currently stored in the CVS system [53] managed by the *Software Release Tool - SRT* [74]. The use of SRT and its possible future replacement is currently under review.

5.2.4 Software Process

No software process is currently used in ATLAS. *Quality Control Group - QC [75]* has been created two years ago, its output has been up to now very limited. Architecture Team is trying to apply the RUP (see 4.2.5) process to its work.

Currently, the ATLAS Software Architecture is under the Review performed by the ad-hoc created *Architecture Review Committee - ARC [77]* . This review, however, resembles only very vaguely standard Software Review process.

Situation with the Software Process in ATLAS is analysed in more details below.

5.3 Software Process

5.3.1 ATLAS Software Process - ASP

The first attempt to apply a Software Proses in ATLAS was the creation of the ASP [78]. While used successfully by several Software Domains in ATLAS (also Graphics Domain), ASP was not really adopted by the ATLAS management and users. There were mainly two objective reasons for that result:

- ASP was not tuned to the HEP environment. In fact, the same problems appeared as those often reported from adoption of RUP (see 4.2.5) and CMM (see 4.2.10) in the software industry. This means, that the problem was not in the adoption of too much from the knowledge of the software world, but too little. The Software Process adoption has stopped in the RUP-like process and hasn't proceeded to the newer processes like XP (see 4.2.7) or FDD (see 4.2.8). This evolution has happened outside HEP, but not yet inside.

- There is a massive opposition to any kind of organized software development in ATLAS from the side of some (key) developers and also management.

We do have a process problem. The problem, however, is not that our processes aren't big enough. The problem is that our processes don't focus on the things that will improve software. [Robert C. Martin]

5.3.2 External Software Handling

ATLAS software depends very heavily on packages originating outside of ATLAS ("external"). Total amount of the code contained in these packages as well as their influence on the design of the ATLAS software will be much bigger that those of ATLAS software. Despite their importance and impact, external packages are not reviewed, so mechanism for accepting or denying them should be defined.

To handle this situation, the guidelines for the handling of the External Software have been formulated as a part of the ASP guidelines. They can be, however, applied independently.

Definitions

External software package: External package is any software package which doesn't originate from inside of ATLAS collaboration. ATLAS people may participate on its development, but the responsible authority should be independent on ATLAS. External package may be reviewed (if authors are willing to) or evaluated.

Software Categories

All ATLAS software packages can be divided into categories using various measures.

Where Is Needed It shows how necessary is to have the package on every machine or every place. It covers the accessibility of the software.

- **Everywhere** Should be available on each machine running ATLAS software. (Example: Control).
- **Nearby** Should be available to everyone, but not necessarily on every machine. Everyone should have access to machine capable of running them. (Example: Calibration).
- **Somewhere** May be missing. Will add additional functionality, which is in general not required but may be useful for specific tasks. (Example: Presentation Graphics).

Replaceability It shows how much is the rest of the ATLAS software dependent on the package and how easy or difficult is to replace it.

- **Core** Is in the core system. Most of other packages heavily depend on it. It is not simple to replace it by other package. It influences heavily design of the rest of the ATLAS software. (Example: Persistent Storage).
- **Replaceable** Can be replaced by other package without big problems. Have little or no impact on the design of the rest. (Example: Mathematics).
- **One-of-many** Several packages with similar functionality exist possibly even at the same time. User may use any of them in a similar way. (Example: GUI).

Impact It shows the impact on the data and results. Some packages may for example make software more user-friendly, but would not change neither the stored data nor the physical results.

- **Data** Have direct impact on the stored (persistent) physical data. (Example: Reconstruction).

Results Have only impact on analyses results. (Example: Analysis).

No Have no direct impact on data or analyses results. The graphical presentation may, for example, change, but numbers shouldn't. Possible error would not infect directly the physical results. (Example: GUI).

Package Evaluation

1. Anyone can ask for evaluation of software package. She also supplies the filled questionnaire together with additional information which may be useful for evaluation and possibly suggestion for emulators (if special technical items should be addressed). The application goes directly to concerned Domain. Domain tries to get evaluation applications for similar (competing) products at the same time, so it can make reasonable comparison.

2. Domain decides if:
   - The package belongs to that Domain.
   - The package can be evaluated as external package. Note, that only packages which have been reviewed or evaluated should be allowed to be used in ATLAS for obtaining results for eventual publication.
   - Which category package belongs to (approximately).

3. Domain then sends the request to Software Steering Group. It may add any relevant additional information and suggest evaluators.

4. Software Steering Group chooses approximately 3 to 5 evaluators and sends them all available documentation and questionnaire. It also suggests them the category of the package. The evaluators will be mostly internal from ATLAS as satisfying ATLAS constraints is important requirement. Some evaluators may be from HEP outside of ATLAS or even from outside of HEP (to be able to judge more technical items).

5. Evaluators perform evaluation of the package (including comparison with similar packages which are available) and send the filled questionnaire with any relevant comments back to Domain. This period may require evaluation and testing.

6. Domain suggests accepting or denying of the package and planed schema of package handling to Software Steering Group. Domain may perform another checks or testing.

7. Software Steering Group decides about package acceptance and schema of package handling.

Evaluation Questionnaire

1. To which categories does it belong ? Make your own judgment, even if categories are already suggested.

2. Is it useful ? Does it satisfy ATLAS requirements ? Does it satisfy any particular requirement (not yet satisfied or badly satisfied) ?

    - In which timescale is it required ? Does it satisfy immediate need or more long-term strategy ? Is it needed now, in medium term, in long term ?

3. What is the price ? The price for the whole ATLAS collaboration is important. Include also HEP manpower.

    - What is the maintenance and support price ? Include also HEP manpower.
    - How are we protected against the raise of price ?
    - How are we protected against licensing changes ?
    - What effort would be needed to develop it ourselves? In ATLAS ? In HEP ? Try to guess the equivalent price.

4. What quality is it ? Is it state-of-the-art of the current software ? How does it compare to similar packages which are available ?

    - Does it use standards ? Whatever is meant as "standard". Does it use any nonstandard feature which is standardized otherwise ? If so, is it justified ?
    - Does it use Open or Proprietary solutions ? Is proprietary solution justified (if used) ?
    - Is it well encapsulated ? Is it not too intrusive ? Should anything be changed in the rest of the ATLAS software to accommodate the package ? Will anything have to be changed if we want in future to withdraw the package ?

5. Does it run on platforms and environments supported by ATLAS ? If not, will it run ? When ? And for which price ?

    - How fast will be port to new versions of OS, compilers,... ?

6. Are there any ATLAS-wide or HEP-wide constrains concerning it ?

    - Is it used by other packages already used by ATLAS ? So we are already using it anyway ?
    - Does it clash with anything already used by ATLAS ? Are we constrained by another (already made) choice ? Could we change the other choice ?
    - Does anything already used by ATLAS offer the same functionality ? Better or worse ? Is it possible to move ?
    - Is there anything else offering the same functionality inside HEP ? Better or worse ? Are we (will we be) constrained by the other package ?

7. Which level of support is available ? HEP, external, no at all ? Does the maintainer officially exist ?

    - How fast will be bug hunting ? Is there official procedure or just "good will" ?
    - May we request package enhancements ? Is the maintainer willing to enhance package according to our requests ? For which price ?
    - May we request package changes ? Is the maintainer willing to make the changes which would modify the package (so other customers may feel unhappy) ? For which price ?
    - May we request port to other platforms and environments ? Or are we depended on the maintainer choice ? Which is the price ?
    - How is future support insured ? What will happen if current maintainer stops existing.
    - Are we the only customers ? Is the package developed for HEP ? For ATLAS ? For scientific world ?
    - Is HEP interesting for the vendor/maintainer ? What percentage of customers are we ? Judge both the money and the importance.
    - What is previous experience with the vendor/maintainer ? Inside ATLAS ? Inside HEP ? Inside academic world ?
    - How big the producer is ?
    - Is it already used in HEP ? With which satisfaction ?

8. Is documentation adequate ? For both developers and end-users ?

- Is there documentation widely available ? Is it in machine readable form. Can we print it ?
- Is there HEP specific documentation ?
- Is API well documented ?

9. Is source available ?
   - Will source be available in case of end of support ?
   - Will source be available in case of maintainer bankruptcy ?
10. Is design available ?
    - Is design available in our supported format ?
11. Is test suite available ?
    - Can we extend it ?
    - Can we include it in our testing schema ?
    - Will it pass our existing tests ? If it substitutes (part of) our existing package. Maybe test should be a little modified (when interface changes) ?
12. Can it be efficiently used in our Release schema ?
    - Will it reside inside or outside of Release structure ? If inside, should we do anything after each package upgrade ourselves to put it into SRT ? If not, how will we handle it ?

The described procedure is formally accepted and it has been used, for example, to define the role of the WIRED Event Display [83] or Expat XML parser.

5.4 Graphics

The development of the ATLAS Graphics Framework has followed standard Software Engineering practices applicable to HEP environment and OO software system. It has, roughly speaking, followed the ASP (5.3.1) and recently some features from XP (4.2.7) are being tested.

The processes went already through several development cycles from User Requirements capture, through Analysis and High Level Design to the Detailed Design and Implementation. At several stages, the system has been successfully reviewed.

Currently, two working versions of the framework exist, one written in C++, another (more recent) in Java. They aim to satisfy the same User Requirements and they are derived from the same Architecture and top level Design. The Java version is currently more functional and is considered as a base line for the future development.

5.4.1 User Requirements

Following User Requirements have been collected from the input of Domain experts, discussions with Users and during the Review Process.

Definitions

Real and Graphical Objects Real objects in the program are mapped to their visual representation, into graphical objects of different kinds, corresponding to different views (see later). The graphical objects correspond to the stored real objects and to the other real objects created from them. Graphical objects can be described as "model" in MVC (Model-View-Controller) [55] system.

Views Different ways of visualizing sets of visual objects. View is generally a window with defined way of displaying graphical objects.

Operations Any actions made in or from the Graphics. Not all operations are applicable to each object or view.

Operations on Graphical Objects Operations only change the visual properties of the graphical objects, the real objects are left intact. Visual operations can be performed on the graphical objects or on the whole views.

Operations on Real Objects Operations on real objects can change the state of these objects, visual properties of corresponding graphical objects can be also changed.

Classification

All higher-level requirements for the ATLAS software are implicitly inherited and not repeated here. This is mainly true for *Control Domain*.

Graphics requirements fall into one of three categories:

General All graphics subdomains should provide the functionality.

Existence At least one graphics subdomain should provide the functionality.

Environment Infrastructure for the functionality should be provided.

Furthermore, the validity of any requirement may be restricted to subdomains of a specific kind (Event Display, Statistics, GUI,...).

Collected User Requirements are available from the Appendix B - **Graphics User Requirements**.

Large subset of the identified requirements has been already satisfied. Details about it are available from the Design handled by the Together CASE Tool.

5.4.2 Architecture

The Graphics Architecture has been created in order to satisfy the captured Requirements best.

The aim of the ATLAS Graphics is to enable visual representation of the objects existing in the ATLAS software. The Design of the ATLAS Graphics is based on the believe that both requirements and graphics software abilities will be very broad at any time and will constantly evolve. The ATLAS Graphics should be able to accommodate all that diversity and change. This can be accomplished only by extreme flexibility and modularity of the core control structure. (Note the similarities with the light methodologies mentioned in 4.2.6, this makes them ideal for the software development in the Domain). The ATLAS Graphics is a part of the full ATLAS software, it covers its graphical components (Histogramming, EventDisplay, GUI,...).

The new ATLAS software is being built using OO approach. This means, that it consists of Objects. Pure Objects (in whatever programing language) are not very user-friendly, they can only be accessed in the programming way. Sometimes, this access can be quite tricky and the core object interface may be very technical for the real access by physicists. So some way how to make access to objects easier and more useful is needed. And some way how to inspect objects and interact with them is needed, too.

In the OO world, this inspection and interaction will be done again via (other) objects. And as observers should not (unintentionally) influence the running of the program, inspection and interaction objects should not be part of the core software. They will represent the original object to the user. A user wants various ways of inspection and interaction with objects, he needs various **Representations**. Those various representations of course share something, they share the original object's **Model**. From the point of view of the relation to the original object, there are several types of Representations:

Event Display (one-to-one) , where each event has its own representation,

Detector Display (one-to-many) , where the same detector description is used for many events,

Histogram Display (many-to-one) , where information about many objects is accumulated in one histogram.

Representations themselves are again objects, they may have their own Representations (**Chain of Representation**).

When several objects are represented in the same way, they share the **Scene**. The Scene is the interface between the program and the user. It is possible, that several kinds of Representations share the same Scene. Scenes can be grouped according to the nature of the Representation they interface. Representations of the same object are interchangeable within the group and they can be (in principle) generated one from another (**Democracy of Scenes**). Scenes are very often implemented by the external packages, which will change and disappear, new packages (and Scenes) will appear. Requirements to visualization are also changing due to mutual interaction between the users and the package abilities and improvement in the physics understanding. This means that the whole system should not depend on Scenes, it should be possible to change Scenes, remove some and add others.

Translated into more technical terms, it can be described in a simplified way like this: For every object which has to be visualized, there should be an additional object (Representations) for each kind of visualization. This Representation should be external to the original object. Representations of the same kind are visualized using another object (or set of objects) - Scene. Scene implements the particular visualization protocol, very often it's just the interface to some external (to ATLAS) application.

The whole Graphics Architecture is derived from this Analysis. The two major Architectural Principles are:

- **The fact, that any object is visualized should not influence the design of that object.**
- **The Architecture of the graphics Framework should not depend on any particular visualization software.**

Those principles have been recently adopted also by the CMS graphics.

Graphics interacts both with the data and with the running program. Graphics consists of a set of **Views** showing geometrical representation of various **Real Objects** via **Graphical Objects**. Operations are performed on the views and contained graphical objects as well as to original real objects. Any real object (which can be any object, candidate for being displayed) has the potential to be displayed. All objects can be displayed in some way, some objects will be displayable in more ways than others.

The Graphics Design and Implementation is evolving so it has at any moment both running version and plans for further development.

The ATLAS Graphics (in Java) currently contains about 200 classes with about 50 000 lines of code. The file containing all sources and compiled code (and also all graphical data) ("jar-file") is about 500 kB big. This code is actively used, developed and maintained. (The C++ framework is used, but frozen. In several places, it is still needed to generate graphics for C++ objects.)

5.4.3 Control Framework

Most users (i.e. developers, interfacing to Graphics) really only need to understand the Architecture as described in the previous section. Only rarely, more detailed understanding of the Design is needed. In such case, this chapter provides only the basic description and for a full understanding, the full Design Documentation and source code is needed. Only Java Design of the Control Framework is described here as it is more advanced and the C++ version is considered obsolete.

The Control structure of the ATLAS Graphics is divided into two packages:

- **ControlInterface** contains just abstract interfaces of the Control.
- **Control** contains default (reference) implementations of the Interfaces and some extensions (like handling of Collections).

ControlInterface

ControlInterface contains just four (abstract) Interfaces defining the Control structure (its Class Diagram is shown on Fig. 5.1) (described naming conventions are used to identify objects at run time):

`Plottable` could be used to make clear that object implements `model()`. Otherwise, naming convention could be used for a `Model`: `<Object name>Model`.

`Model` can be implemented either by the class with the name `<Object name>Model` or by any class given by the `<Plottable>.model()` method. In such cases, it gives customized features to Reps and Scenes. `DefaultModel` defines simplest `Model` implementation, it could be extended.

`Rep` is a representation of an object (possibly `Plottable`) suitable for a `Scene`.

`Scene` represents a kind of graphical View (usually a way of representing real objects in a graphical package. "Graphical" has very broad meaning here - it can describe also files,... `Scene` will try to get the proper `Model` for getting proper `Rep`. If proper `Rep` doesn't exist, some Scenes can use default behavior, which doesn't require `Rep` and works just with `Plottable` (`Object`) itself. For such `Scene`, the Model-Rep creation steps can be skipped even when the proper `Rep` exists (by call to `setDefaultBehaviour()`) and default behavior can be used directly (this will be generally much faster). There are three possibilities:

- `Scene` accepts any `Object`: This would be convenient for default behavior, where it's not necessary to create `Model` and `Rep`. It would also make `Scene` usable without the Graphics Framework. It would be, however, difficult to implement collaboration of Scenes, which is performed via Models.
- `Scene` accepts `Model`.
- `Scene` accepts `Rep`: This would not allow default behavior.

`Scene` accepts `Model` and asks it for the proper `Rep`. If it gets `null` and default behavior is allowed, it asks it for `Plottable` and uses it. Otherwise, it doesn't performs any display operations (but it may add `Model` into its container). It may display information.

Control

Control package contains implementation of the ControlInterface interfaces (the Design Diagrams are shown on Fig. 5.2, 5.3, and 5.4):

`DefaultModel` gives default interface of any `<Object>`.

`DefaultScene` can be used as a prototype for any `Scene`. It implements methods which can be re-used among various Scenes. It does it usually in the simplest way to fulfill the `Scene` (and `CollectionScene)` contract. For CollectionModels, `DefaultScene` just loops over contained Models.

`Modeller` is a *Factory* which maps any object into its `Model` (its functioning is apparent from the Collaboration Diagram shown on Fig. 5.5):

1. if `Model` for the `Plottable` already exists, it is used,
2. if `Object` implements from `Plottable`, its `model()` is used,
3. if `<Object>Model` class exists, it is used,
4. if `Object` implements `Collection`, `CollectionModel` is created,
5. if no special `Model` exists, the `DefaultModel` one is used.

Soft References to `Model` are stored not to interfere with Garbage Collector.

`Representer` maps `Model` into its `Rep` corresponding to the supplied `Scene` (its functioning is apparent from the Collaboration Diagram shown on Fig. 5.6):

1. `Model` may overwrite call to `rep(...)` so that `Representer` is not used,
2. if `<Scene><Object>` exists, it is returned (it may exist in the `<Object package>Graphics` or in the `<Object package>` and names of all superclasses of `<Scene>` are checked),
3. otherwise, `null` is returned and if `Scene.allowsDefaultBehaviour() == true`, default behavior is used. `Representer` doesn't check if suitable `Rep` already exists - this is task for `Scene`.

`SceneList` coordinates collaboration between Scenes and Objects. It contains list of Scenes and list of Models of Objects (Plottables) displayed on them. It calls proper methods of `Scene`. `For CollectionModel`, proper `Scene` functions are used.

`register(...)` registers `Scene`

`unregister(...)` unregisters `Scene`

`add(...)` adds `Object`

`display(...)` adds `Object` and displays it on all registered Scenes

`remove(...)` removes `Object`

`update(...)` updates `Object`

`CollectionModel` models object which implements `Collection`. It contains `Collection` (`ArrayList`) of Models modeling objects from object `Collection`. `CollectionModel` is handled by `SceneList` and `CollectionScene`, but `Scene` can override default behavior (which just loops over contained Models). Also `CollectionModel` can be, naturally, overwritten.

`CollectionScene` is an Interface which can handle CollectionModels in a specific way.

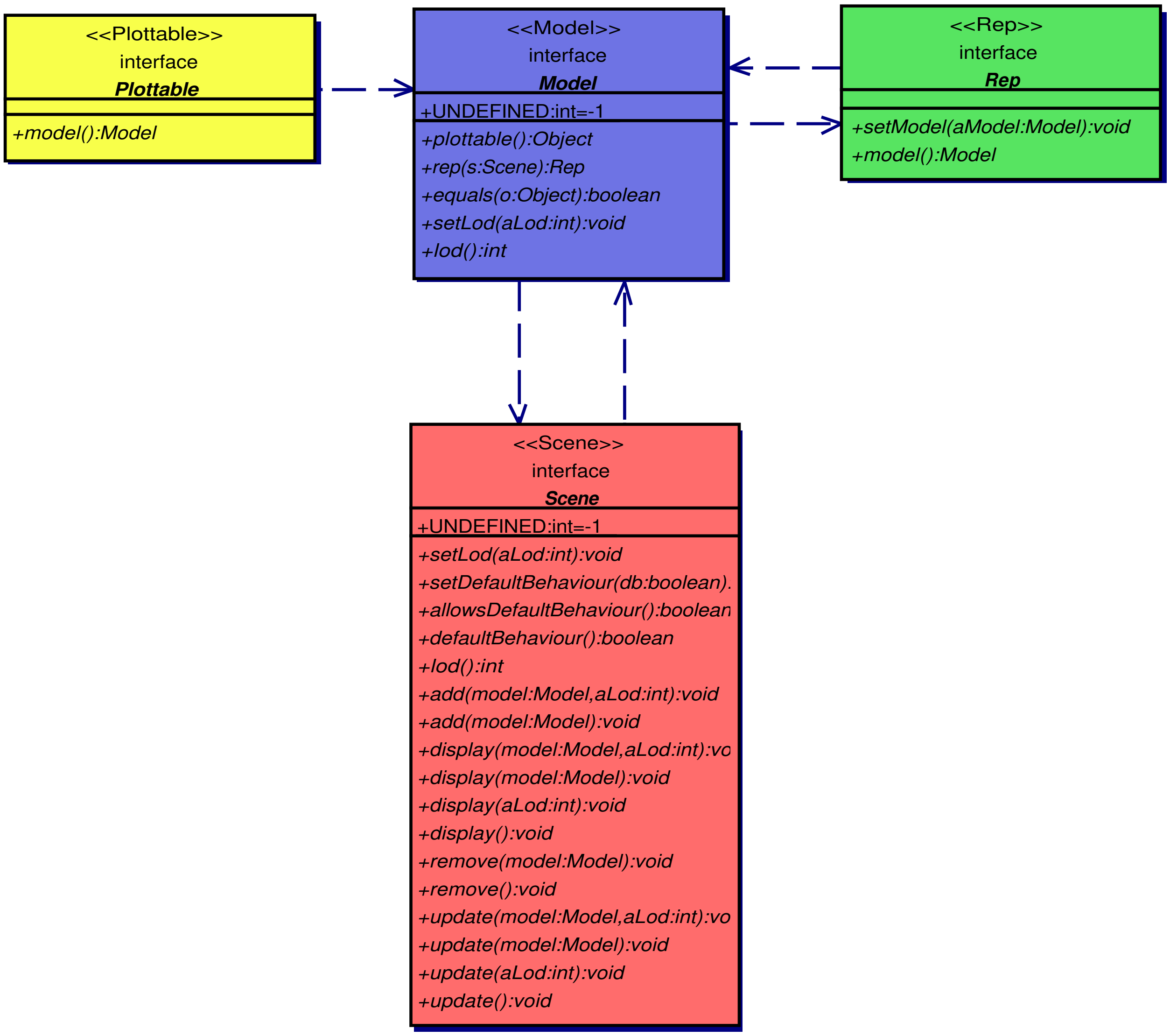

Figure 5.1: Class Diagram of the **Graphics Control Interface**. Colors are used to describe stereotype: yellow for `Plottable`, blue for `Model`, red for `Scene`, green for `Rep`.

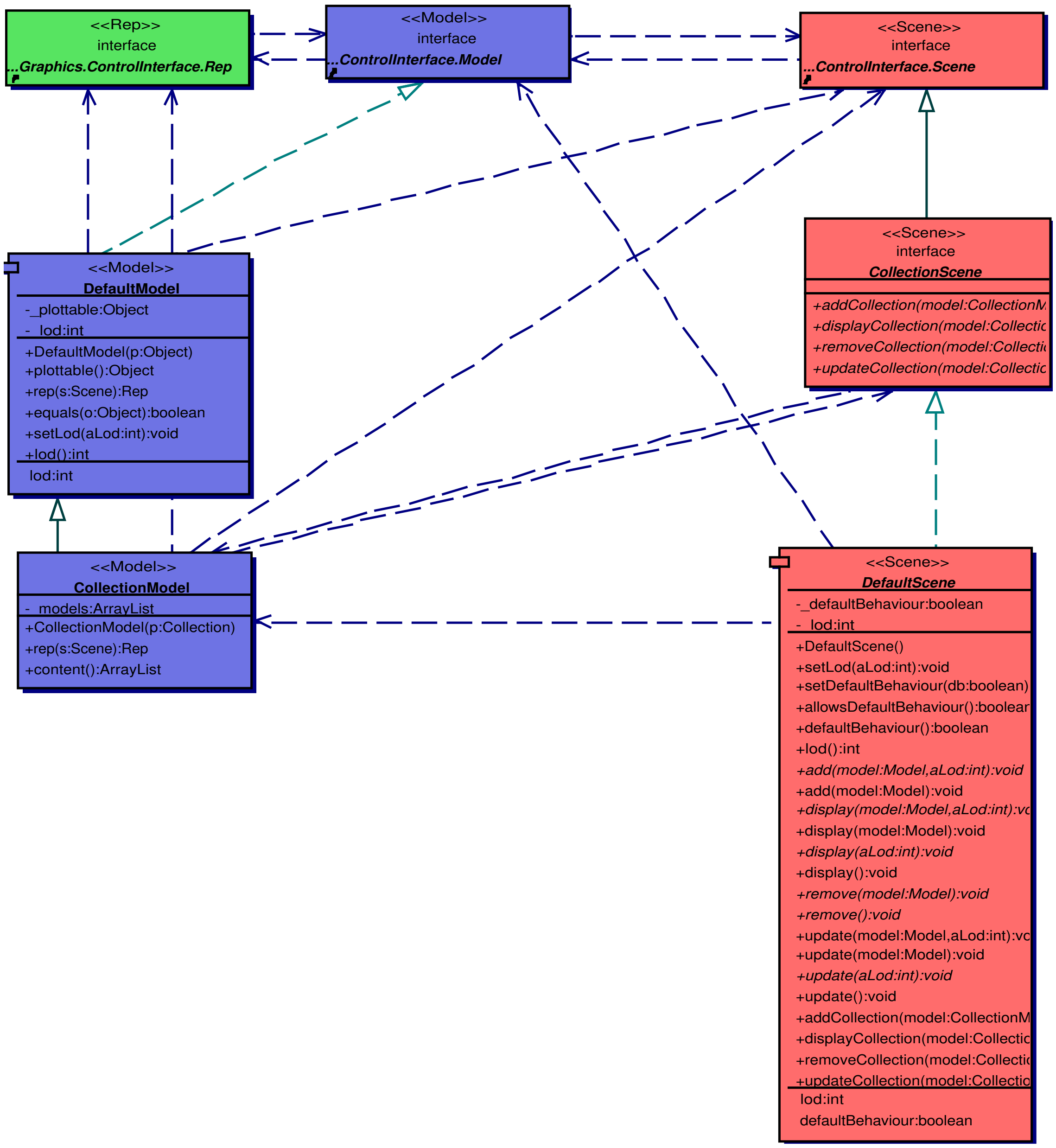

Figure 5.2: Class Diagram of the **Graphics Control**.

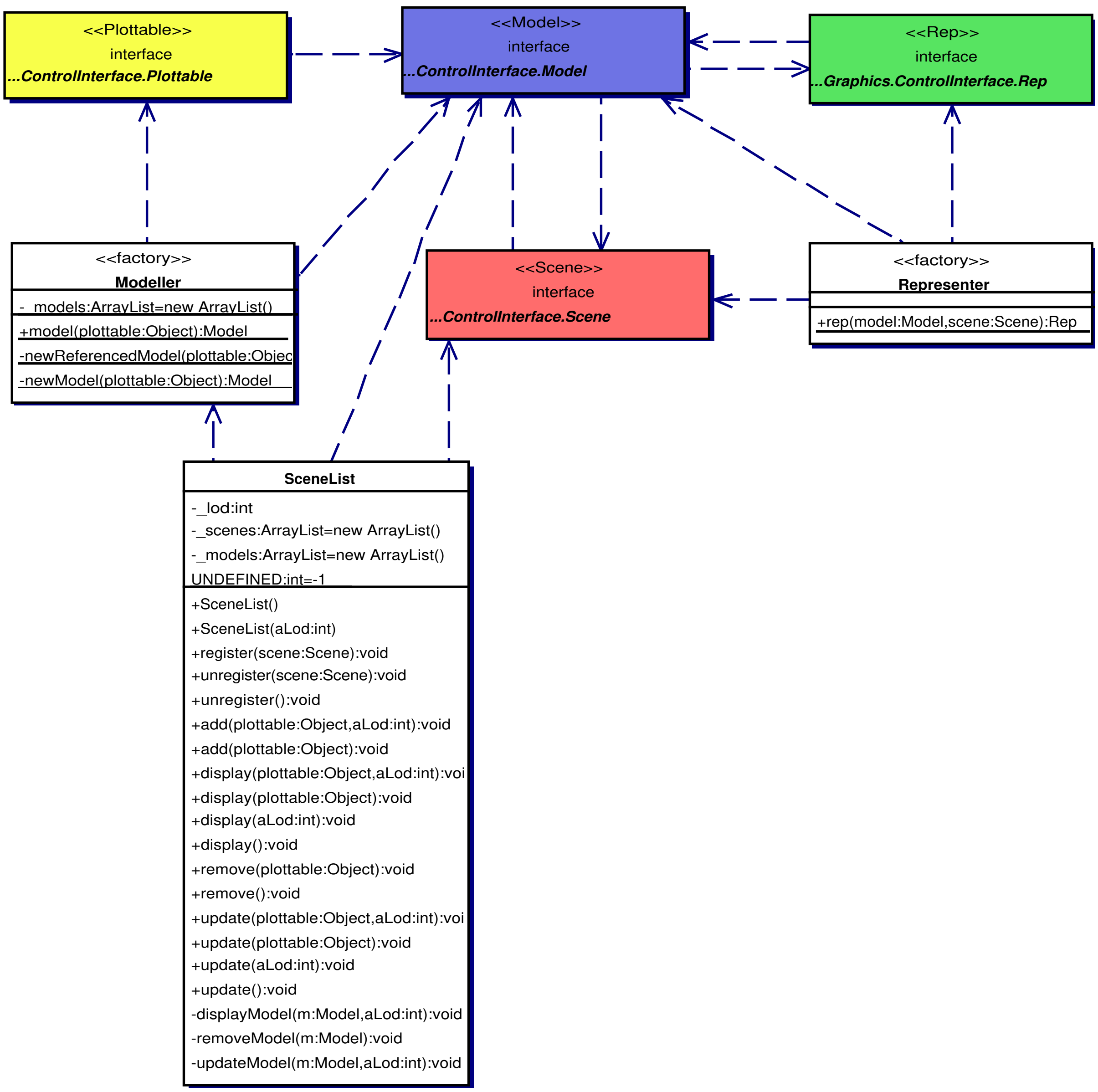

Figure 5.3: Class Diagram of the **Graphics Factories** (`Modeller` and `Representer`).

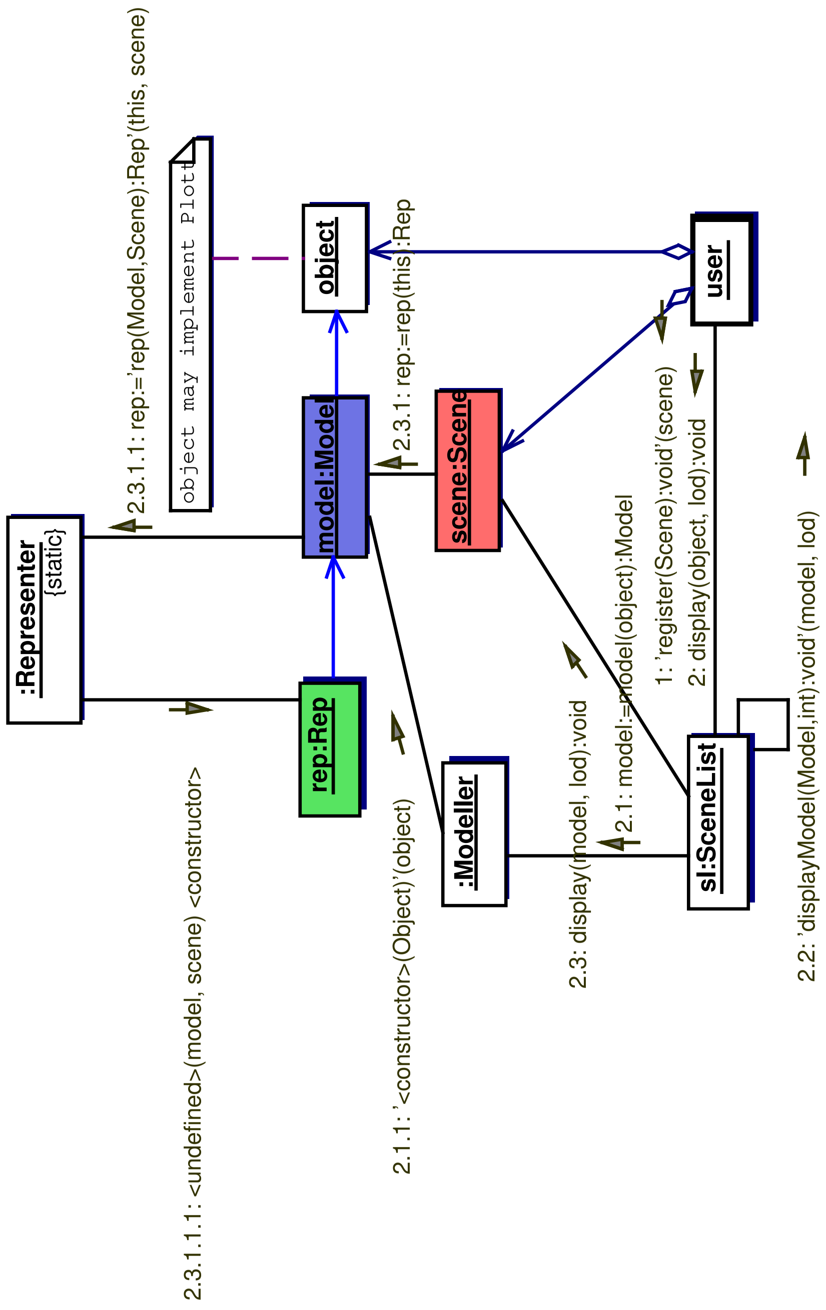

Figure 5.4: Collaboration Diagram for the **Display of an Object**.

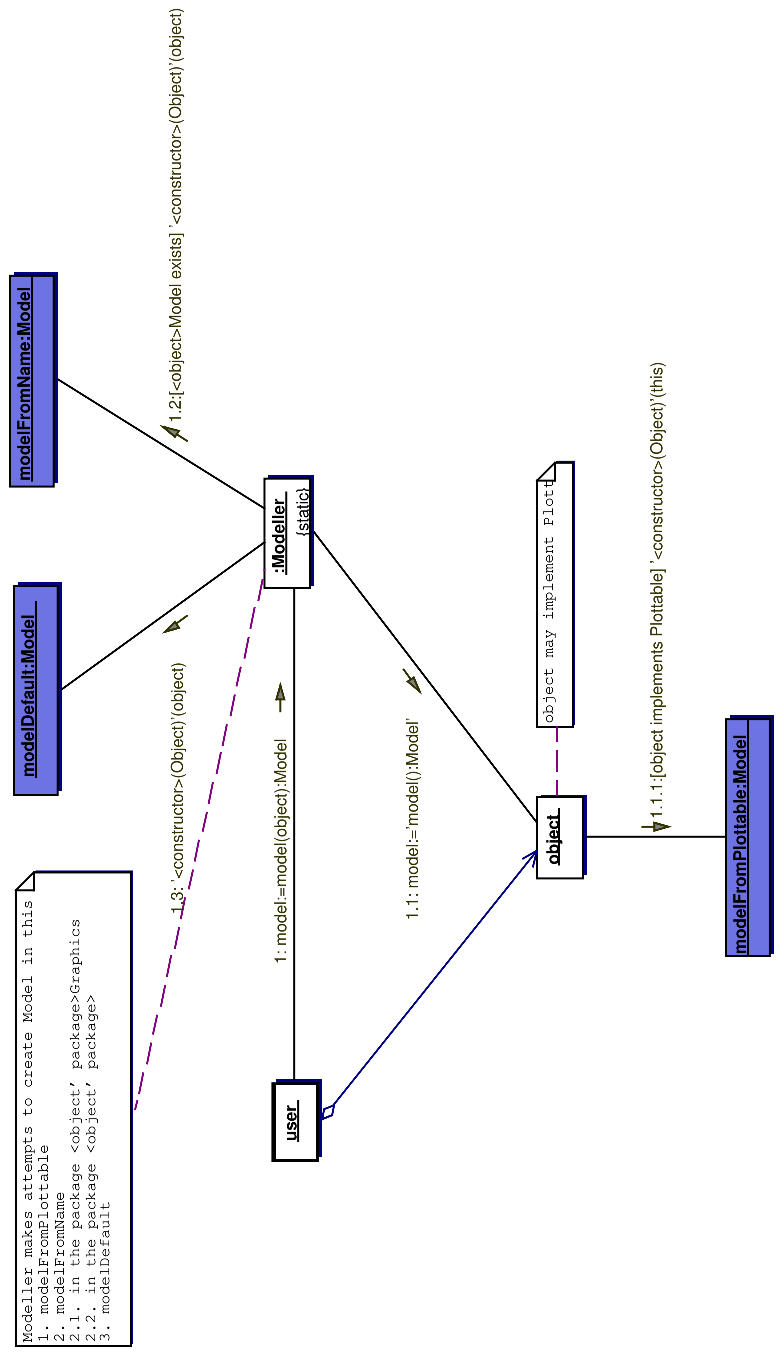

Figure 5.5: Collaboration Diagram for the **Creation of a Model for an Object**.

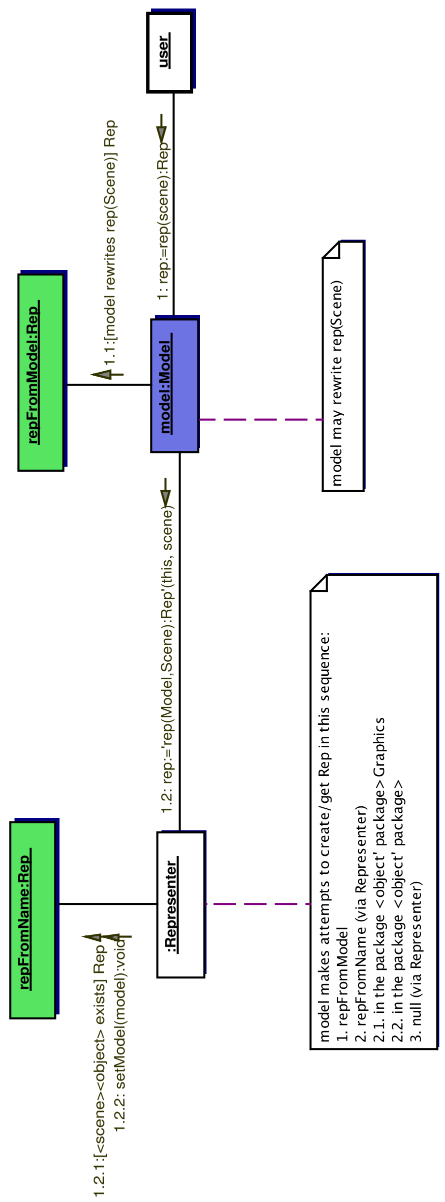

Figure 5.6: Collaboration Diagram for the **Creation of a Rep for a Model**.

5.4.4 Scenes

Scenes represent the ways of visualizing real objects (and interacting with them). In agreement with the Graphics Architectural principles (see 5.4.2), Scenes will come and go. In the following sections, the main existing Scenes are described.

Text [79]

Text Scene creates Text representations of an objects and sends them to the output stream. Default operation (for objects without Representation) is possible using Reflection mechanism (so object's features can be asked at run time).

Text Scene is the simplest Scene, it is useful for debugging purposes.

XML [80]

XML Scene creates XML representations of objects and sends them to the output stream. Several XML dtds are supported allowing for different views of the data. Default operation (for objects without Representation) is possible using Reflection mechanism.

Created XML files for some representation can be used as input (format) to other programs. For example, *atlasevent.dtd* [83] defines the Event structure understandable by several Event Display programs like GraXML, Atlantis/Jive or WIRED.

Aravis [81]

Aravis is a simple interactive wire-frame graphics system. It is based on the graphics part of Arve [71], but has been substantially re-implemented to interface it to the rest of the ATLAS graphics software. Aravis is integrated with the C++ Graphics Framework and will be replaced once the migration into Java is completed.

Atlantis [82], Jive

Atlantis is an Event Display with operations suitable for detailed study of full Events. It is currently the most sophisticated Event Display in ATLAS. It allows wide range of graphical operations on event and offers special projections (like *Fish-eye*, *Clock-transformation* or *V-plot*). Atlantis has been created adopting Aleph [2] Event Display program *Dali*.

Currently, Atlantis (written in Fortran) is being rewritten in Java under the (provisional) name Jive. (Example of running Atlantis is shown on Fig. 5.7, running prototype of Jive is on Fig. 5.8.)

WIRED [83]

WIRED is a framework for building HEP Event Displays which can be used interactively across the Internet using World-Wide Web. An Event Display written using the WIRED framework enables users of an HEP collaboration to explore, scan and research events in remote places using ordinary WWW browsers. WIRED offers functionality of the similar kind as Atlantis (is, however, currently less functional).

WIRED is used also by other experiments (Delphi [9], BaBar [10], Chorus [11],..).

WIRED is one of the foundation applications of the HEP Studio (described in 5.4.5). (Running WIRED can be seen from Fig.5.9.)

Object Browser [85]

Object Browser provides visual access to any Java Object (including its members). It visually resembles standard file manager, where objects are represented as files and collections as directories.

Default operation (for objects without Representation) is possible using Reflection mechanism. **Context Sensitive Actions** will be added, possibility of **Drag-and-Drop** interoperation with other Scenes is under evaluation.

Reusable parts of Object Browser are being migrated into FreeHEP library and HEP Studio (described in 5.4.5). (Two snapshots of running Object Browser are on Fig. 5.10 and 5.11.)

JAS

JAS Scene is just an interface to the **Java Analysis Studio [86]** program.

JAS itself is the tool aimed at the easy, intuitive and powerful analysis of the HEP data using Java language. It offers both graphical interactive interface for the visual operations (rebinning, scaling, cuts, free slices and projections, ...) and programing interface (in Java) for writing analysis code. JAS is easily extensible, new data sources, plotting subsystems and analyzing code can be added by the users. The tool is built using modern OO techniques which makes the use of all of its features transparent.

JAS is one of the foundation applications of the HEP Studio. (Example of running JAS is shown on Fig. 5.12.)

AIDA [87]

AIDA Scene allows to create AIDA (*Abstract Interfaces for Data Analysis*) [88] histograms. Created histograms can be written into text or xml file. XML file can be read by standard Analysis programs like JAS.

Test [89]

Test Scene offers help in testing components of the Graphics Framework.

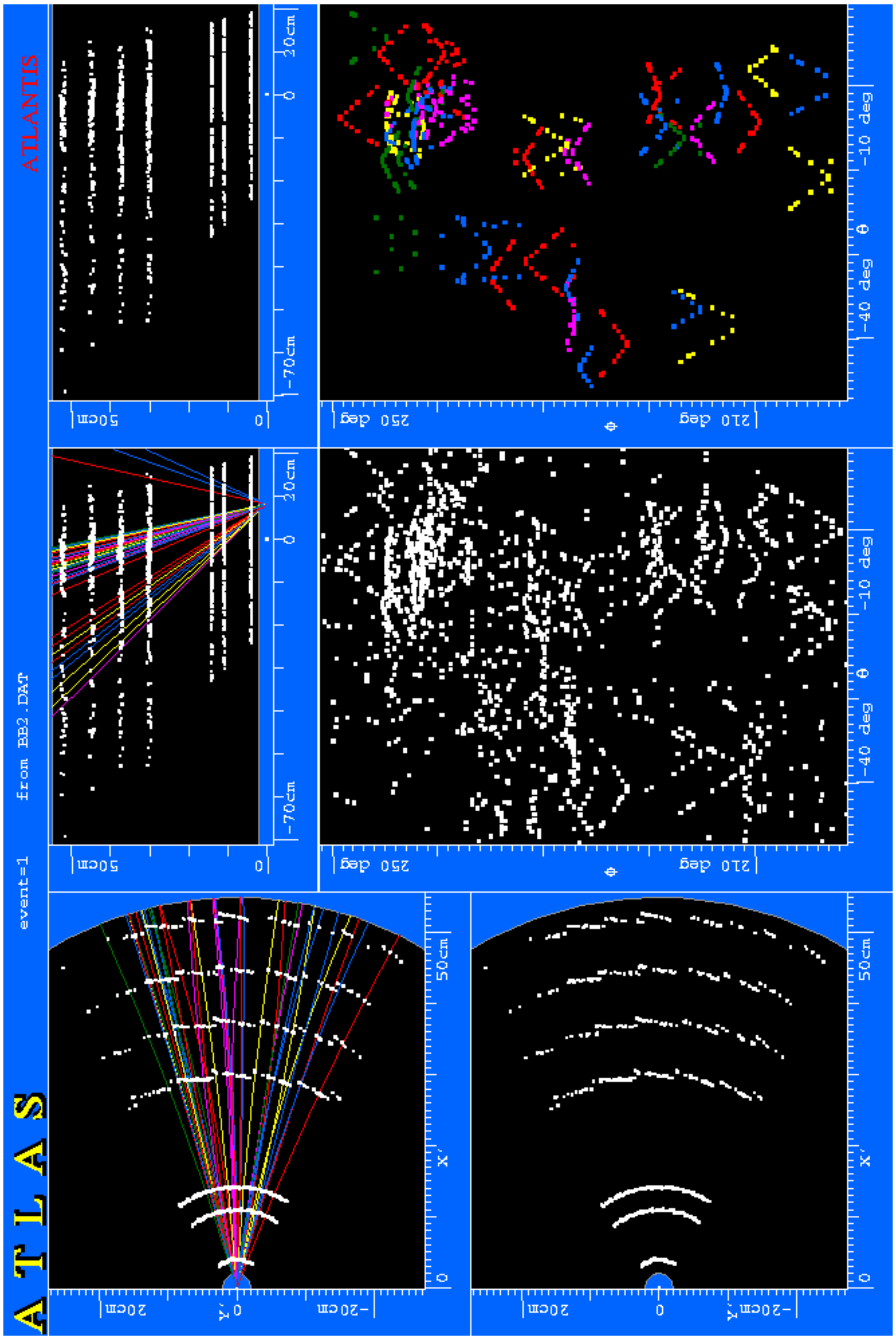

Figure 5.7: Typical Atlantis View.

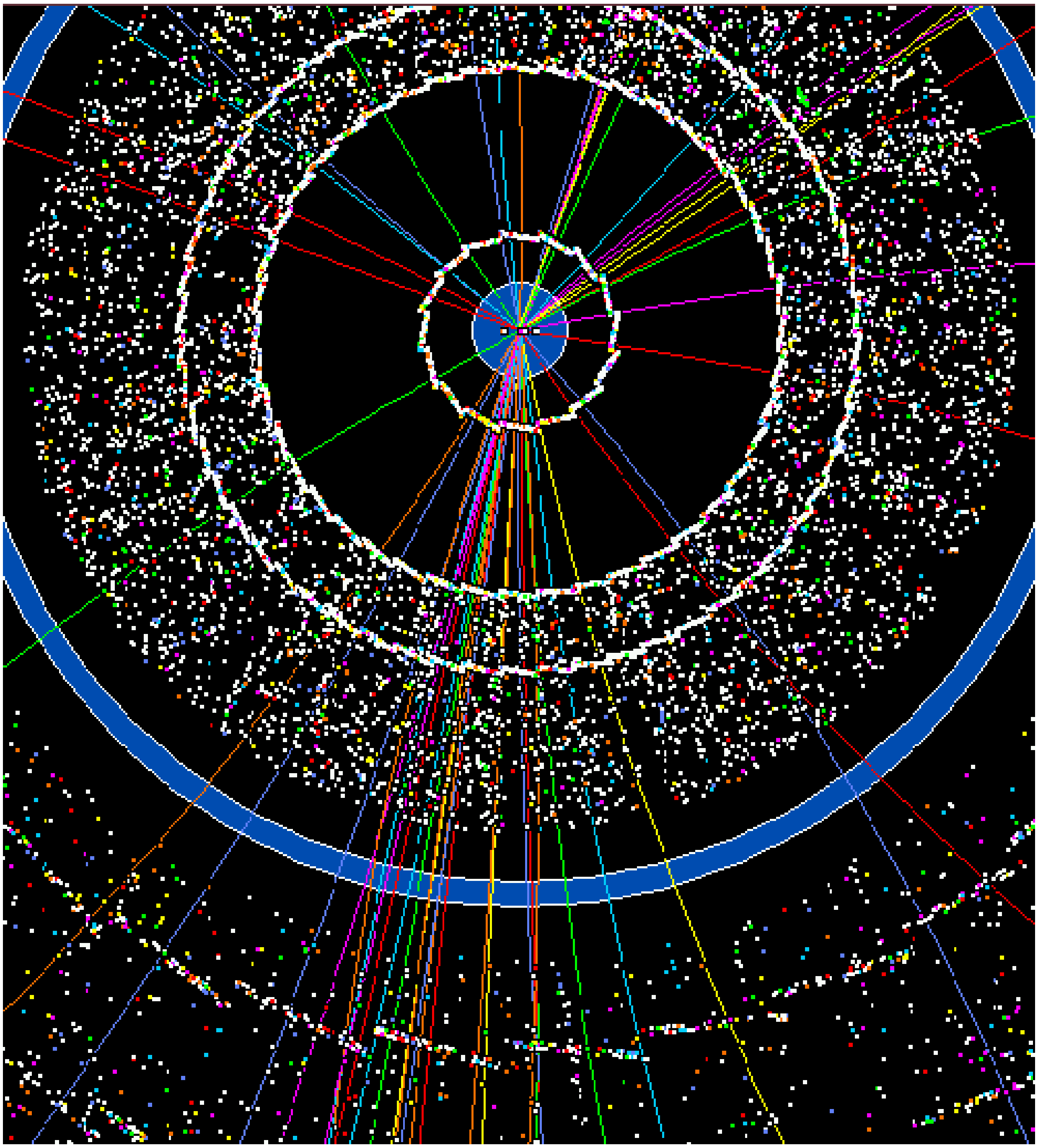

Figure 5.8: Jive View.

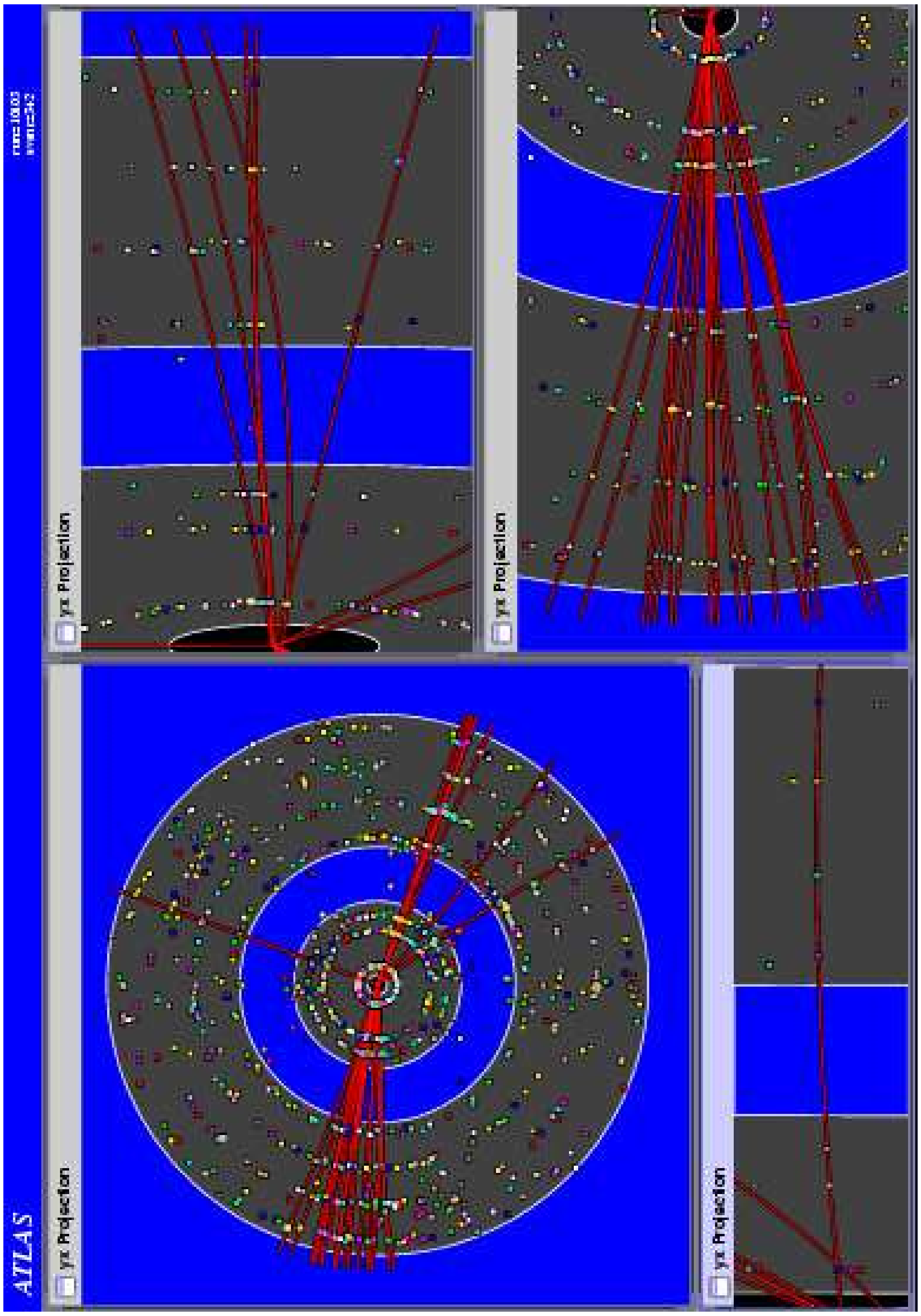

Figure 5.9: Typical WIRED View.

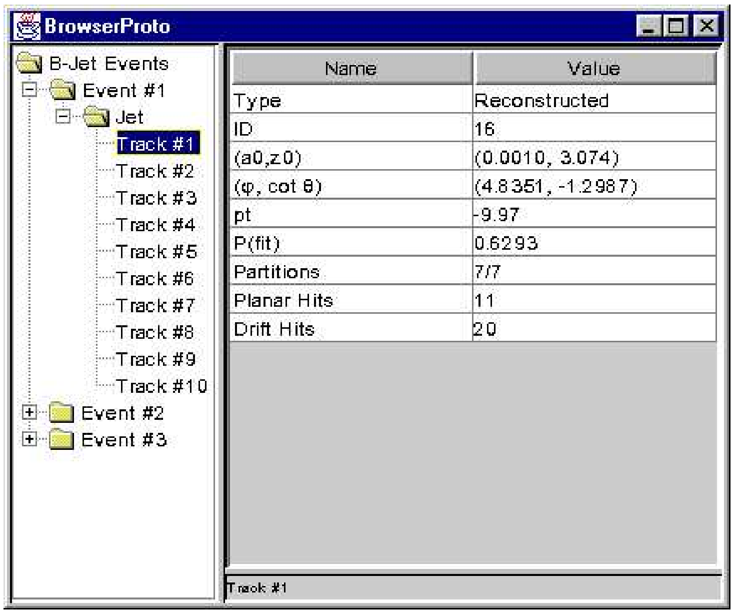

Figure 5.10: Typical ObjectBrowser View.

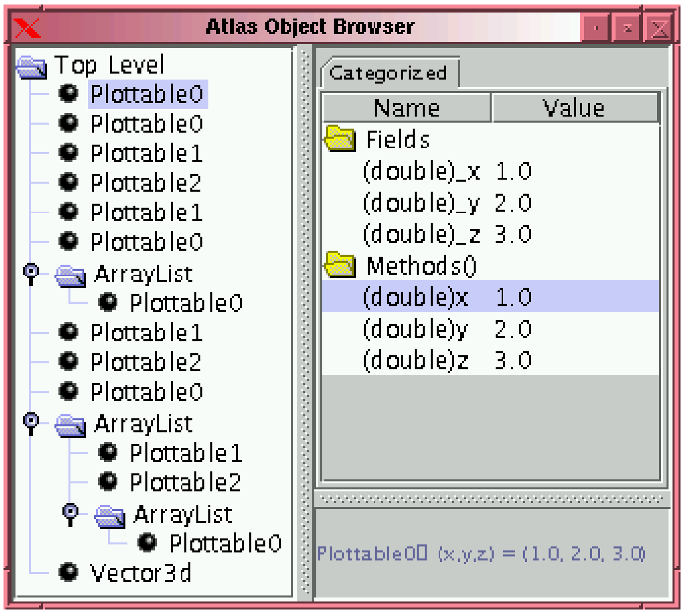

Figure 5.11: Typical ObjectBrowser View.

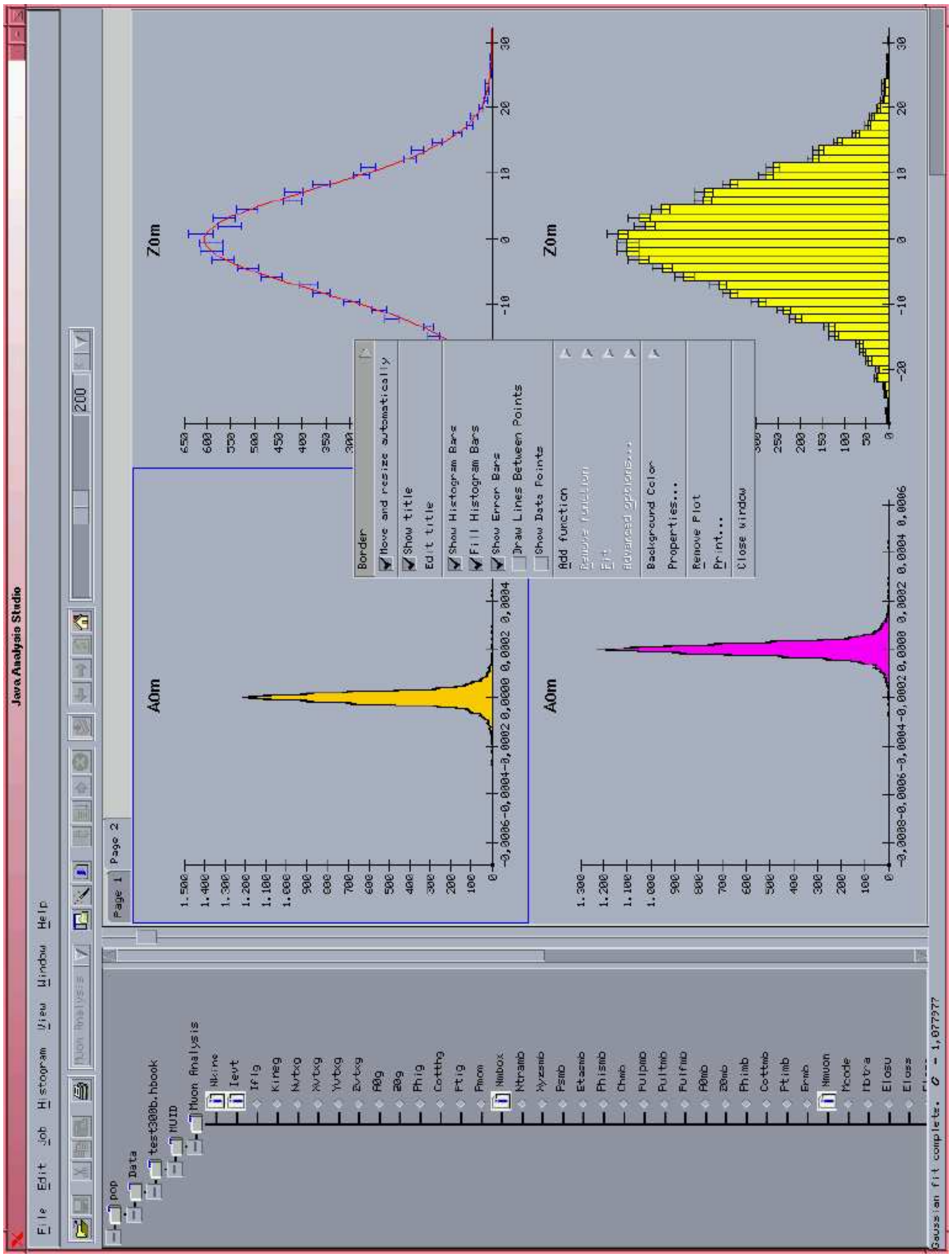

Figure 5.12: Typical JAS View.

GraXML [84]

GraXML is a program for the 3D visualization of the ATLAS Event and Detector data from the ATLAS offline software and AGDD [64] and Event XML files. It uses Java3D [54] to create, render and manipulate 3D Scene-Graph. Modular architecture allows easy extensions of the supported objects, currently all ATLAS objects and most of the ALICE and Geant4 detector description objects are supported.

(Examples of running GraXML with various kinds of visible objects are shown on Fig. 5.17, 5.18, 5.19 and 5.20.)

GraXML, while still being prototype, offers full functionality for browsing and manipulating 3D data. All the major User Requirements(5.4.1) functionality is available:

- Standard 3D visual operations, like Rotation, Translation, Zooming, Scaling, Sheering and Skewing. Those operations can be anizotropic, if it is applicable. All those operations can be applied either to the whole Scene (*Global Operations*) or to the selected (picked) objects (*Local Operations*). Local operation allows, for example, to move one object away from others. Operations are available via Drag mouse action, using keyboard or through the menu. "Go to" operation is also available, it moves observer close to the selected object (or away from it).
- Changing of visual properties of an object. The color of an object, its transparency properties and its polygon status (solid - wireframe - vertexframe) can be changed interactively.
- Both Parallel and Perspective projections.
- Cutting the 3D Scene from front and from back.
- Selective switching of the objects on and off from the hierarchical Tree Navigator (shown on Fig. 5.16).
- Snapshot saving into the jpg file (PostScript and Scalable Vector Graphics - SVG will follow).
- Context sensitive help.
- Optimized or fully Interactive navigation. The Optimized mode uses much smaller memory footprint and is therefore useful for the large 3D Scenes (more than tens of thousands of simple objects). Interactive features are limited in this mode.

- Quality level. Higher quality levels offer features like better antialiasing, but requires significantly more CPU.

- Output to VRML [48]. The functionality of writing a Java3D picture into a VRML file is unique and apparently no other application, which can provide this functionality successfully, exists. VRML output can be also translated into a POV [36] Raytracing format file, which can be then used to create photographic-quality pictures (incl. reflections,...).

- Context sensitive introspection and actions allowing inspecting and calling features of the real objects connected to the 3D objects. This means, that for example, user can re-fit a Track by interacting with its graphical representation. This functionality, however, requires collaboration of the back-end (reconstruction) framework, which doesn't take place yet.

The top-level design of the GraXML package is shown on the Fig. 5.13.

Several reusable components have been implemented during the development of the GraXML program, which are now becoming a part of the FreeHEP library and HEP Studio:

- Java3D representation of Geant4 [57] solids. All basic solids are implemented and present in FreeHEP library. See Fig. 5.14 for an example of such object.

- Mechanism of the creating Tree View Navigator for selective switching on/off (groups) of Java3D objects. Several in-memory tree databases are created to facilitate this mechanism. See Fig. 5.15 for the connections of various trees used in the mechanism and Fig. 5.16 for the example of the Tree Navigator.

- Interacter mechanism allowing for dressing of naked Java3D objects with local interactivness and Contexts Sensitive Actions.

5.4.5 FreeHEP and HEP Studio

All potentially reusable components of the ATLAS Graphics Framework are (or will be) part of the HEP-wide Java library - FreeHEP [90]. ATLAS contribution to the FreeHEP library currently consists of GraXML components (Java3D implementation of Geant4 solids has been already contributed) and application-neutral Object Browser.

Two of the key Scenes which are used in the Framework, WIRED and JAS, are the foundation of the HEPStudio - *Analysis Framework created from* *independent modules*. Other components from the ATLAS Graphics will be included in future - Object Browser, GraXML and Atlantis/Jive.

5.4.6 Java-C++ Interoperation

Most of the ATLAS Graphics Scenes and the complete Graphics Framework are implemented in Java. Java is the language of choice for the ATLAS Graphics. It is quite certain that Java will be used as the implementation language of the overall ATLAS software as well. However, currently ATLAS software is written in mixture of C++, Fortran77, Fortran90, Java and proprietary language Age. As the core Framework (Athena) is written in C++, seamless interoperation between Java and C++ is a necessary ingredient of the ATLAS Graphics strategy.

Java-C++ interoperability for ATLAS Graphics Framework is being solved using **Java Access to C++ Object - JACO** package. JACO allows automatic creation of all interface code necessary to access C++ objects from Java. JACO is a part of the FreeHEP library.

More information about the role of Java in ATLAS is available from the Appendix A - **On the importance of Java to ATLAS**.

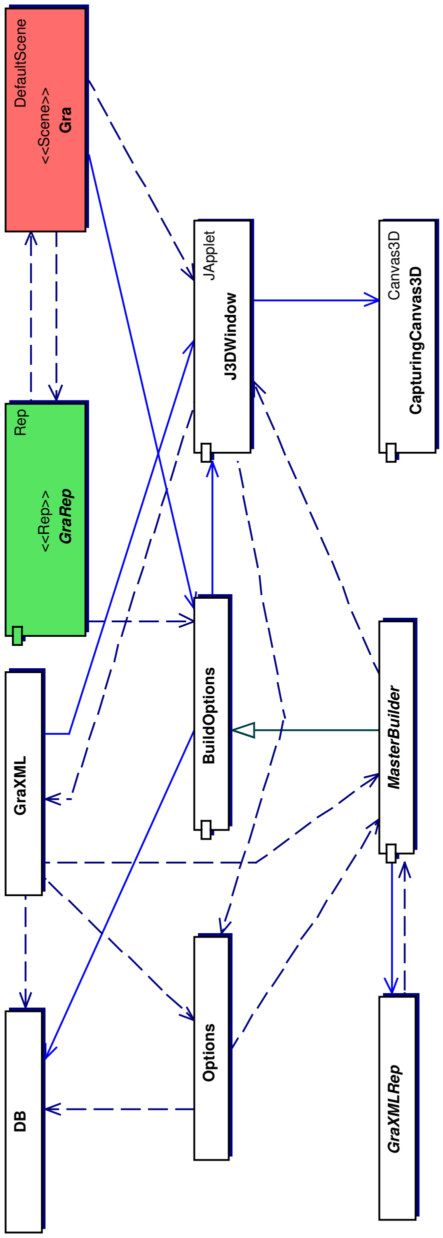

Figure 5.13: Top level Class Diagram of the **GraXML**. `GraXML` handles start and environment setup, using `Options`. `DB` handles on-the-fly created database for parsing complex XML files, building SceneGraphs and navigation. `MasterBuilder` is a *Factory* which handles the creation of the SceneGraph taking into account guidelines defined via `BuildOptions`. It also mediates access to XML files and their parsing. Most of the real work is delegated to the `Builder` specific to the XML dtd. `GraXMLRep` is a abstract supperclass for all *Factories*, which create Java3D elements. There is one such *Factory* per `Plottable` object or XML element type. `J3DWindow` creates and organizes the *Graphical User Interface*. `CapturingCanvas` allows to save a snapshot of the SceneGraph into a file. `Gra` and `GraRep` are standard Graphics classes implementing the `Scene` and the `Rep`.

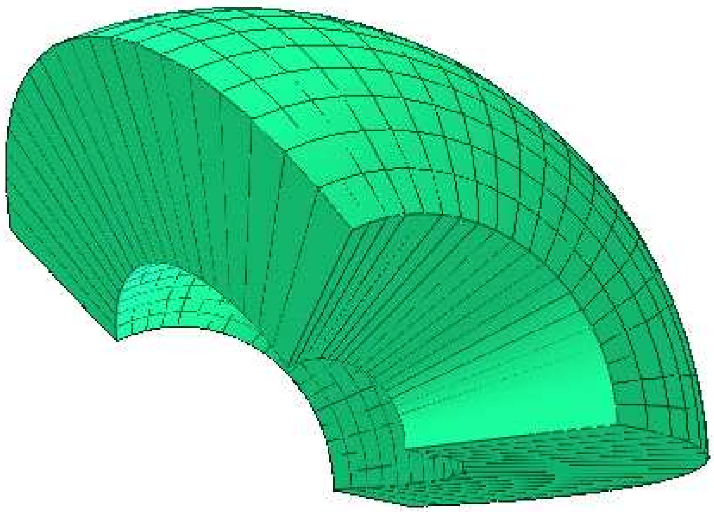

Figure 5.14: Example of the Java3D implementation of the Geant4 Solid. `SphereSegment` is shown which is extended to Geant4-implementing `G4Sphere`.

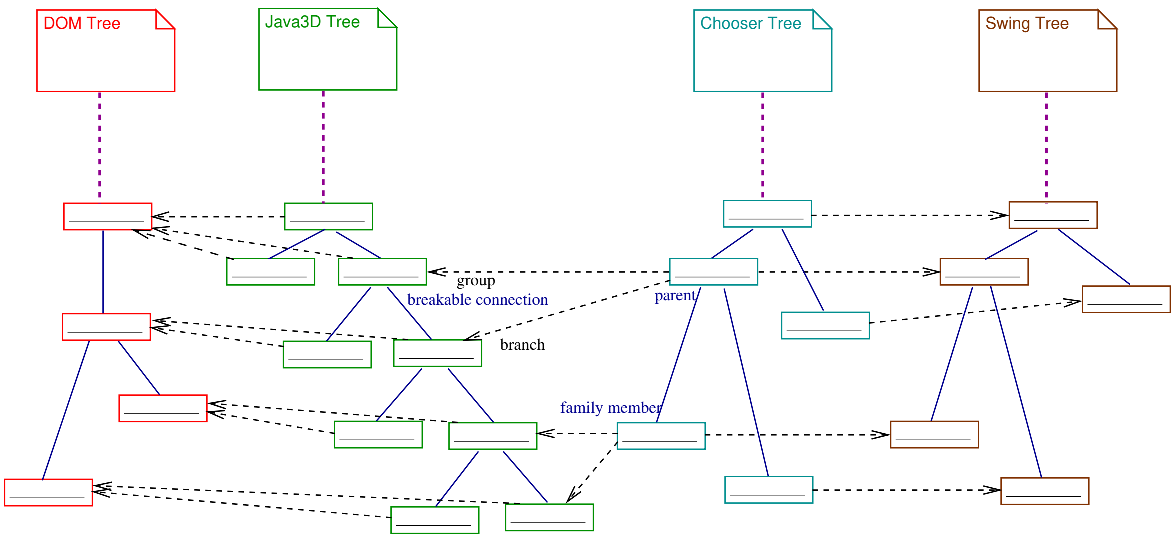

Figure 5.15: Object Diagram of the Navigation chain through the tree object structure of shown objects. `DOM Tree` contains the snapshot of the parsed XML file or tree of the objects obtained from the other (reconstruction) program. `Java3D Tree` is its representation using Java3D objects. One element from the DOM Tree is generally mapped into subtree of the Java3D Tree. `Chooser Tree` contains all information needed for the navigation in the Java3D Tree, it also mediates a list of breakable connection in it. `Swing Tree` forms the context of the Tree View Navigator.

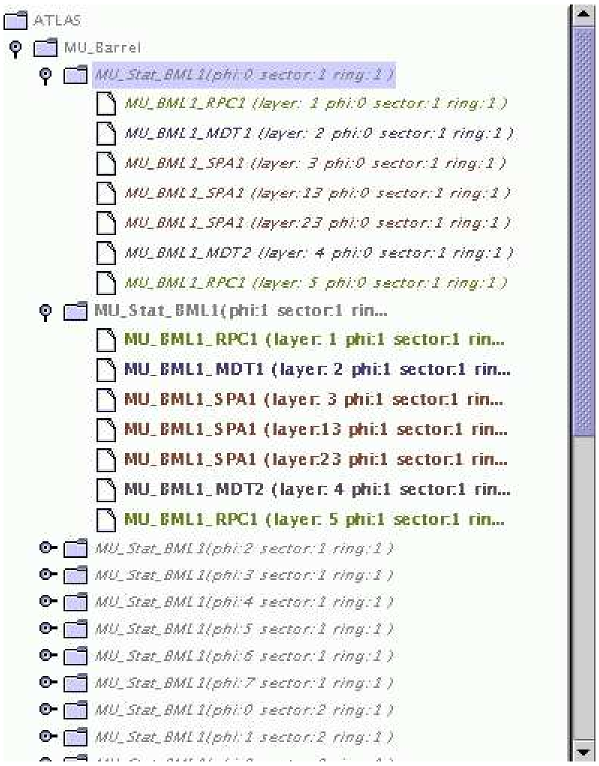

Figure 5.16: Example of a *Tree Navigator*. Elements (and folders) shown in bold letters are visible, elements (and folders) shown in italics are invisible, folders shown in normal font are partly visible. User can show and hide elements from this Tree Navigator.

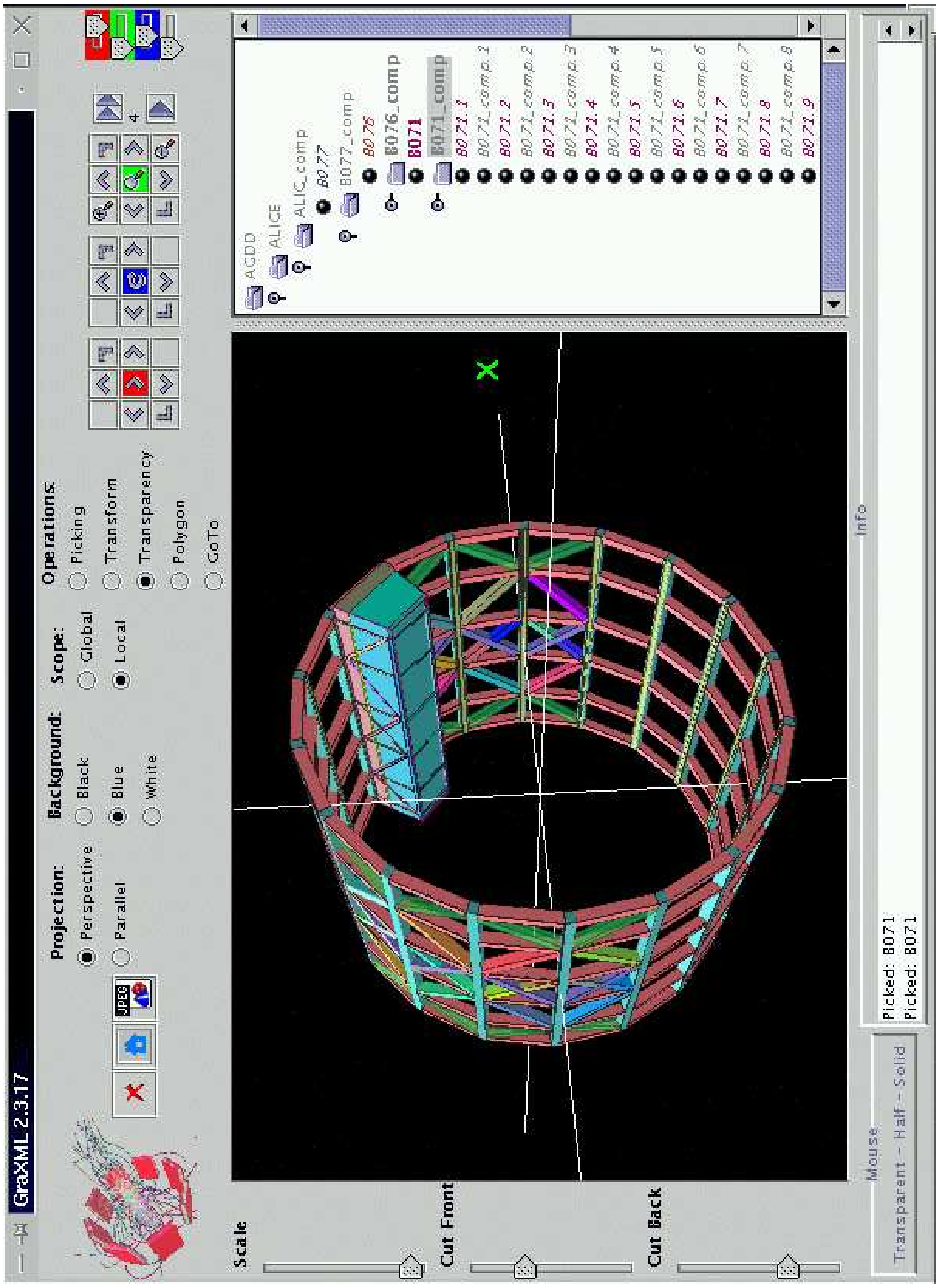

Figure 5.17: Standard GraXML View. Part of the ALICE detector is shown, which demonstrates the generality of the GraXML program.

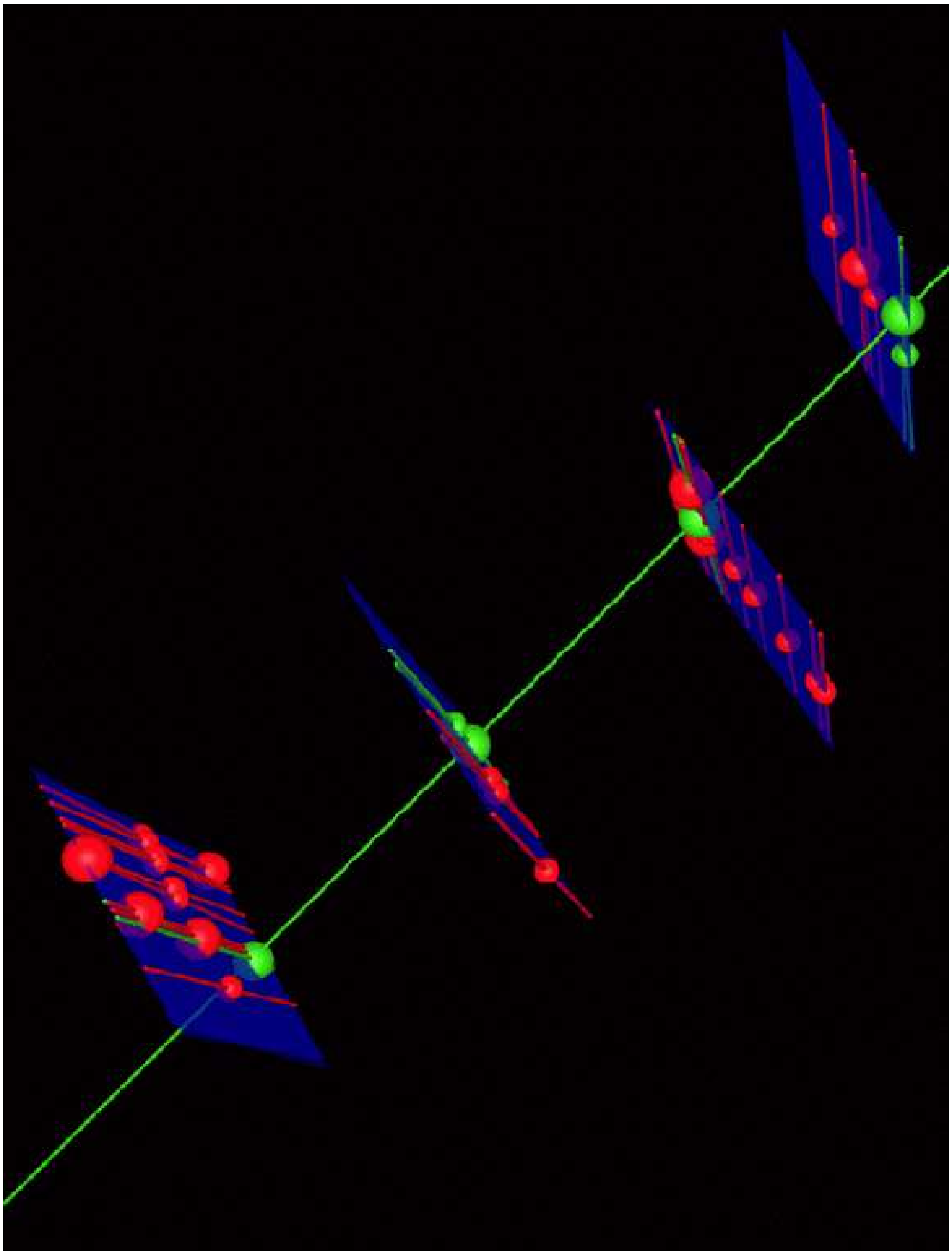

Figure 5.18: Detailed View of a Single Track in ATLAS SCT Detector shown in GraXML.

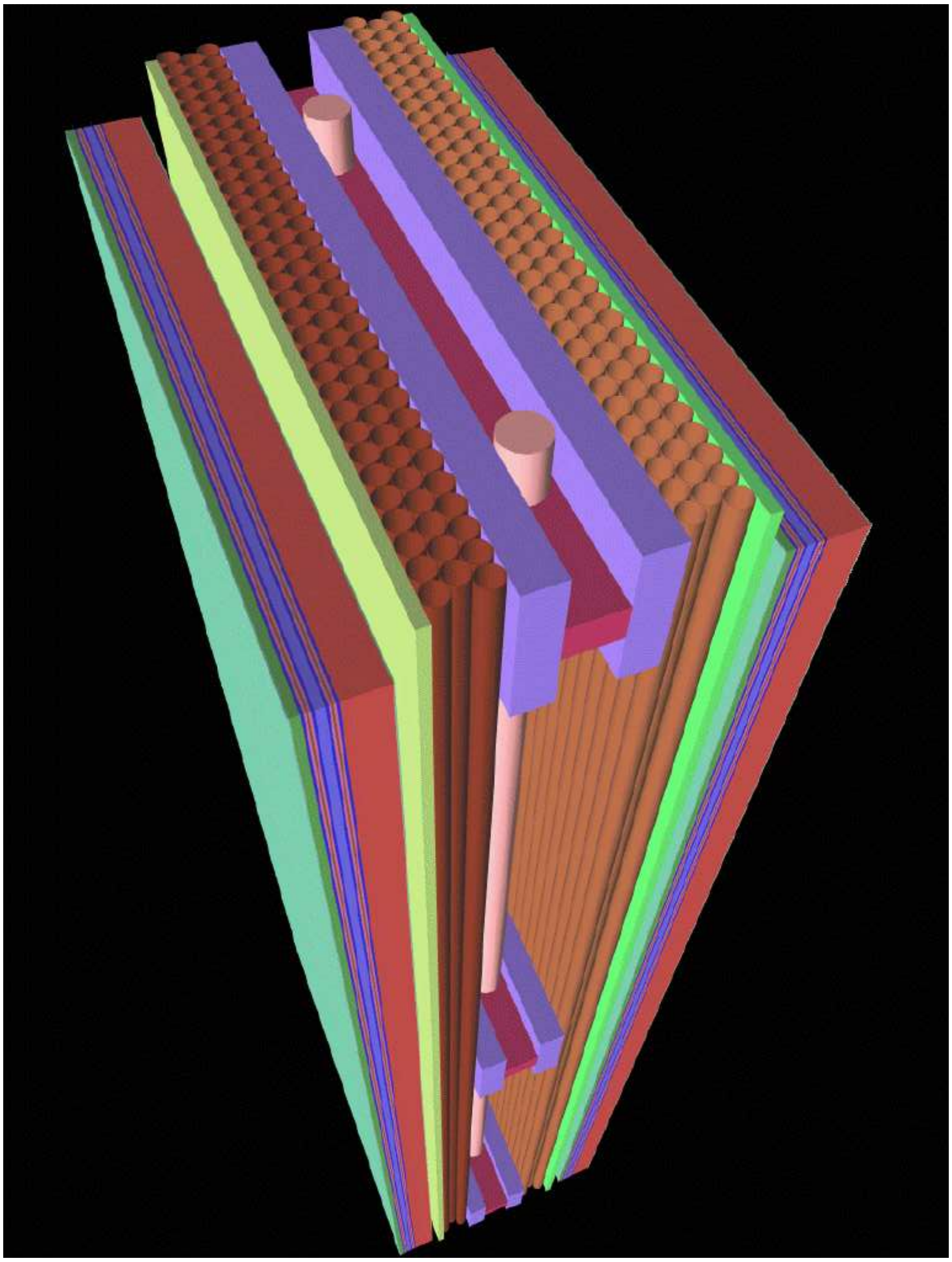

Figure 5.19: Geometry of one ATLAS Muon Chamber shown in GraXML.

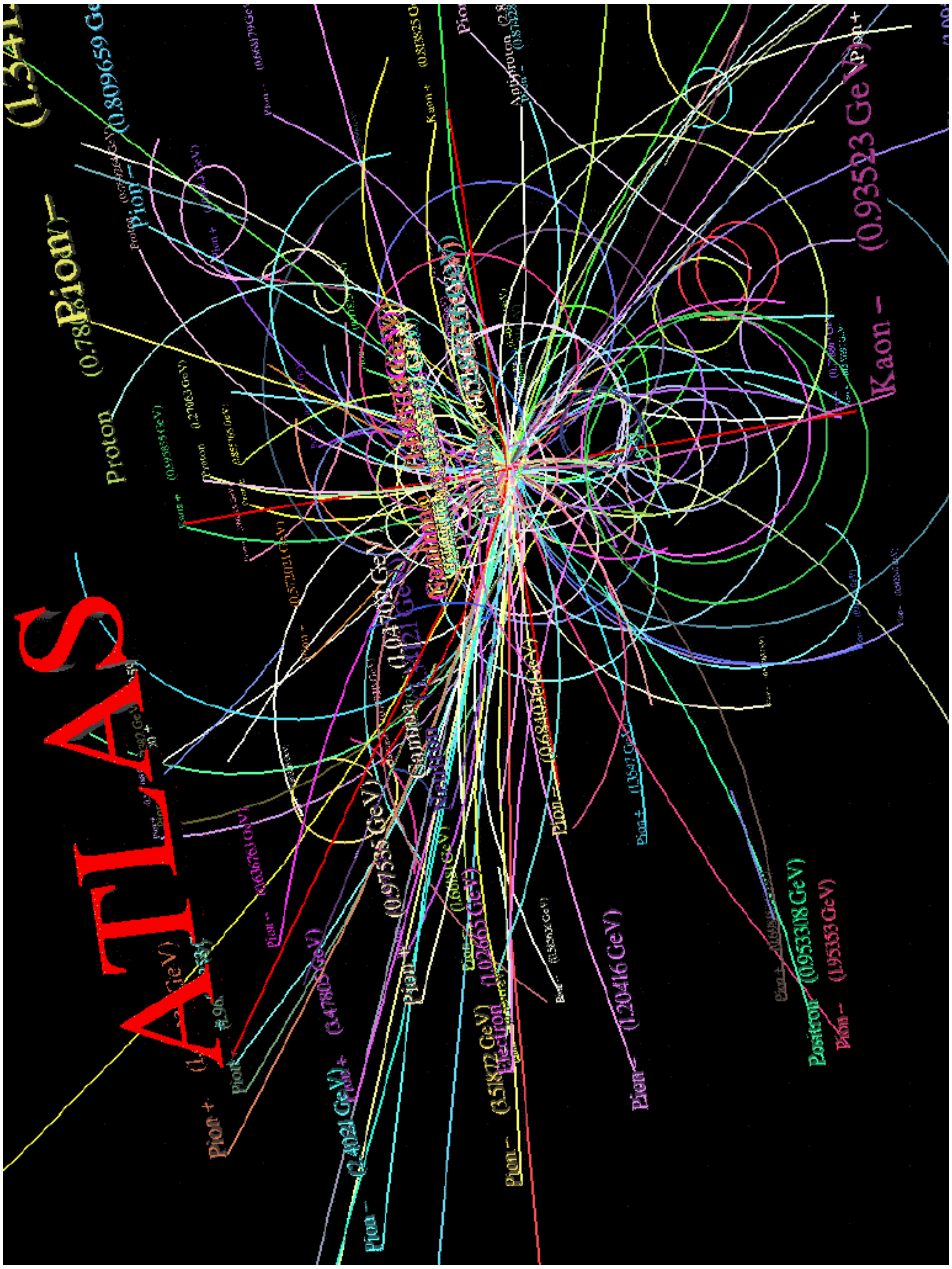

Figure 5.20: Tracks of one Event shown in GraXML.

5.5 OPATREC

OPATREC is a general algorithm for Global Pattern Recognition in Inner Tracker. It finds all Hits (= space points) from the Road (= set of Hits) belonging to each Track (= circle).

It is rather difficult to find non-straight lines (circles, spirals) from several their points. OPATREC algorithm changes this problem to a problem of the clustering (in higher dimension space). Clustering algorithms are simpler and widely available. The main stages of the algorithm are:

1. Create Road by giving it all its Hits: $\rho = r, r \in R^3$
2. Find (all) 3-tuples among Hits from the Road: $c_{ijk} = (r_i, r_j, r_k) \in R^{3\times 3}$
3. Put a circles through 3-tuples. $R_{ijk} = (r_0, x_0, y_0, t_0, a_0) \in R^5$
4. Create space from the parameters of circles.
5. Clusterize in the created space with a defined metric (create Tracks): $(R, w)_I = (\{R_{ijk}\}, w_{ijk})$
6. Find first approximation to p_t from the circle curvature: $p_t = 1/r_0$
7. Recall all Hits belonging to clusters (Tracks): $\{r_i\}$
8. Call fit with recalled Hits

Here, points 2. $\div$ 7. belong directly to the responsibilities of the class Road, others are either given to the Road by the creating Class or performed by objects of other classes (possibly by inheritance). Demonstration of the algorithm' function is shown on Fig. 5.21 and 5.22.

A prototype of the OPATREC algorithm has been implemented in Fortran77 and Eiffel languages.

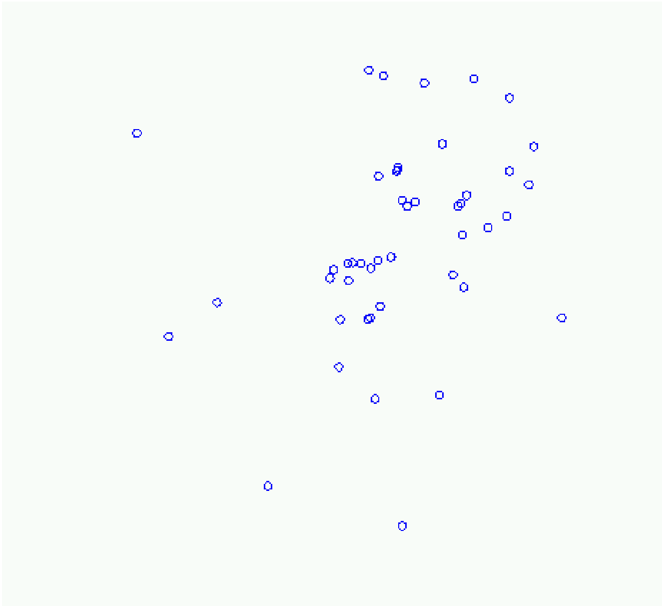

Figure 5.21: Set of artificial hits before application of the OPATREC algorithm.

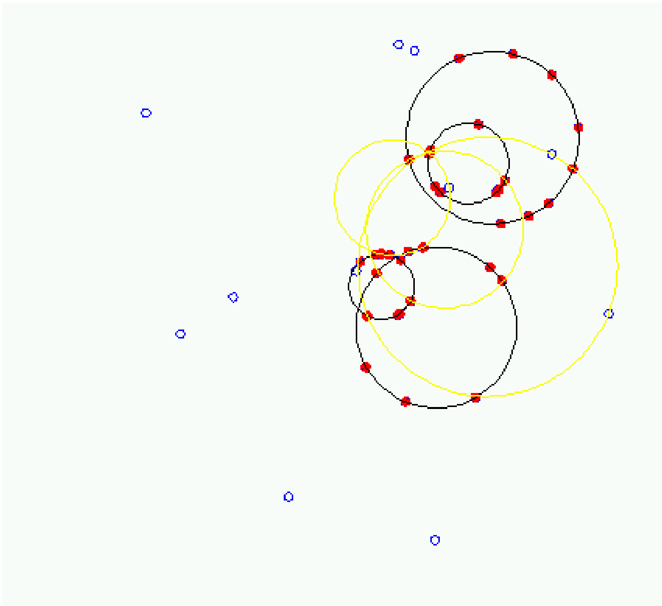

Figure 5.22: Set of artificial hits with found tracks after application of the OPATREC algorithm.

Chapter 6

Physics

This Chapter covers the Physics topics. After a short introduction to the ATLAS Physics and B-quark Physics in general, two topics are brought in. Section 6.3 describes the work done for the Evaluation of the CP Asymmetry Dilution Factor dependency on the ATLAS detector setup. While this topic doesn't bring any new physics theme, the results of it have been used in the formation of the decision between the various design choices for the ATLAS detector setup. Section 6.4 contains original contribution about the possibility of beauty baryon polarization measurement in ATLAS. The gained results have been published [92] and are also incorporated in the ATLAS Detector and Physics Technical Design Report [14]. The results in this Theses are updated with respect to published results to take into account new experimental data. Author has performed all the study described in Sections 6.3 and 6.4 and the simulation work described in general part of the Chapter in collaboration with his colleagues from the Prague ATLAS group, the Section 6.3 together with M.Smižanská, the Section 6.4 together with M.Smižanská and R.Lednický.

6.1 ATLAS Physics

Fig. 6.1 shows some important pp production cross-sections and interaction rates as a function of the center-of-mass energy $\sqrt{s}$. The total cross-section is estimated to be between 90 mb and 130 mb. The total luminosity will be about 10^5 $\mathrm{pb}^{-1}\mathrm{year}^{-1}$.

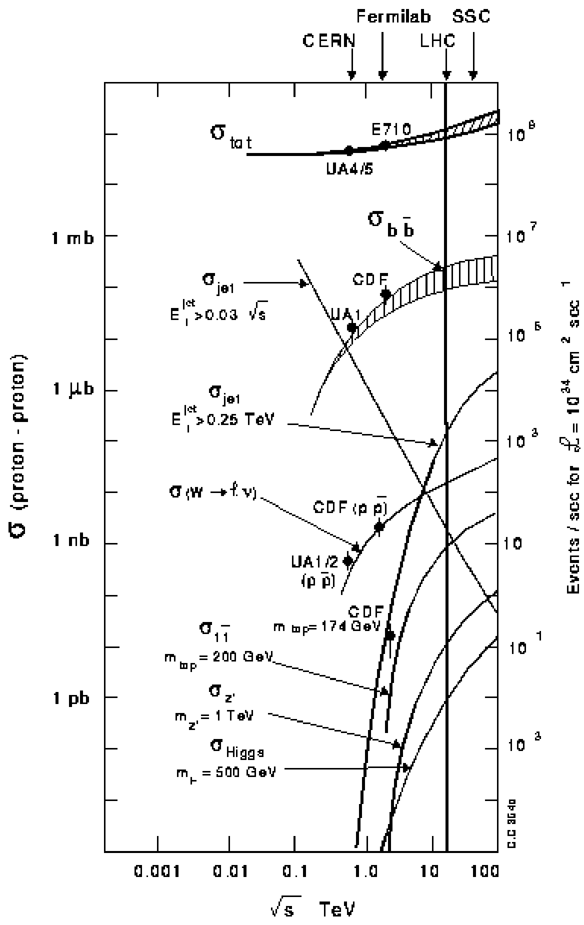

Figure 6.1: The cross-sections and rates of some characteristic processes dependence on $\sqrt{s}$.

One of the main goals of LHC is to study the mechanism of the symmetry breaking in the electroweak sector of the *Standard Model*, which is associated with the nature of the Higgs mechanism, the existence of the Higgs particle and the behavior of the cross-section involving the gauge bosons $W^{\pm}$ and Z. Besides the Higgs, also the physics of the B-quark and Top quark production and decay will be studied.

The physics potential of the ATLAS is described in the *ATLAS Detector and Physics Performance Technical Design Report* [14]. The main topics are summarized in this section, B-physics is covered later.

6.1.1 Standard Model Higgs

The most prominent benchmark processes for the LHC detectors are the different Higgs-decay channels for the Higgs-mass from 80 GeV/c^2 to 1 TeV/c^2. The Higgs coupling to other particles is proportional to their masses. It couples to quarks and leptons of mass m_f with a strength $g\frac{m_f}{2M_W}$. If the Higgs mass is very large, the coupling of the Higgs to itself and to longitudinally polarized gauge bosons become large. Requiring that these coupling are weak enough (so perturbation theory is applicable) implies that $M_H \leq 1$ TeV/c^2. In other words, the width of the Higgs boson is proportional to the cube of its mass (for $M_H < 2M_Z$) and that a boson of mass 1 TeV/c^2 has a width of 500 GeV/c^2 and hence dissolves into background. The lower limit of the Higgs mass is about 114 GeV/c^2 (as obtained by LEP [1]). The *Minimal Super-symmetric extension of the Standard Model* predicts an upper limit for the lightest super-symmetric Higgs boson of 130 GeV/c^2.

Higgs Production

The production cross-section of the Higgs is about 1 pb, which gives about 10^5 produced Higgs bosons per year. The cross-section steeply increases for a low-mass Higgs. The most dominant production mechanisms are the gluon-gluon fusion and the $WW(ZZ)$ fusion. Higgs are also produced by the $\bar{t}t$ fusion and W or Z bremsstrahlung.

Higgs Decay

Only a few decay channels of the Higgs boson are accessible to the experimental observation since the decay channels have either small branching ratios, or they are lost in a large background. Statistics will be the limiting factor in the detection of the Higgs boson, in particular for Higgs with a large mass. The possible open channels are:

$M_H <$ **120 GeV**/c^2 The Higgs decays dominantly into quarks ($H \to b\bar{b}$), which channel is unfortunately lost in the QCD background. The channel $WH \to l\nu\gamma\gamma$ has a negligible background, but also the production rate. The third candidate decay is the $H \to \gamma\gamma$ with again enormous background. All these backgrounds can be reduced with excellent photon resolution and γ/jet and $\gamma/e^{\pm}$ discrimination. Hence an excellent performance of the electromagnetic calorimentry is required.

120 GeV/c^2 < M_H < 800 GeV/c^2 The Higgs decays as $H \rightarrow ZZ^* \rightarrow l^+l^-l^+l^-$ with clear signature. The main background is coming from prompt muons, decay muons, hadronic punch-through, neutrons and muon induced electromagnetic secondaries so the good geometrical and kinematic acceptance acceptance for leptons is crucial. For large Higgs masses the signal becomes rate limited due to the growing Higgs width. The accelerator luminosity starts then to be more important then the detector performance.

M_H > 800 GeV/c^2 The channels $H \rightarrow ZZ \rightarrow l^+l^-\nu^+\nu^-$ and $H \rightarrow WW, ZZ \rightarrow l^{\pm}\nu 2jets, 2l^{\pm}2jets$ are frequent and can be detected using high-p_T leptons and high missing p_T measurements.

The Higgs sensitivity of ATLAS is shown on the Fig. 6.2.

6.1.2 Minimal Super-Symmetric Extension of the Standard Model - MSSM

The simplest extension has two Higgs doublets. The physical spectrum contains one charged Higgs boson $H^{\pm}$, two neutral scalars h, H and one pseudoscalar A. For the top quark mass $m_{top} = 174$ GeV/c^2 (and other plausible constrains), there is a bound $M_h < 130$ GeV/c^2. The search requires efficient secondary vertex detection for τ leptons and b-quarks and high-resolution calorimetry for jets and missing transverse energy.

6.1.3 Super-Symmetric Searches - SUSY

If SUSY particles exist, they will be produced and detected at the LHC for masses up to 2 TeV/c^2. The lightest SUSY particle is stable, but almost non-interacting so the search sets strict requirement on the missing E_T capability of the detector.

6.1.4 Top Physics

The $\bar{t}t$ cross-section for LHC is about 1 nb leading a very high statistics of top-quarks so the top-mas resolution will be limited by the theoretical uncertainties more then statistics. It will be possible to perform detailed study of its decays channels like $t \rightarrow bH^+$ or $t \rightarrow Zc$.

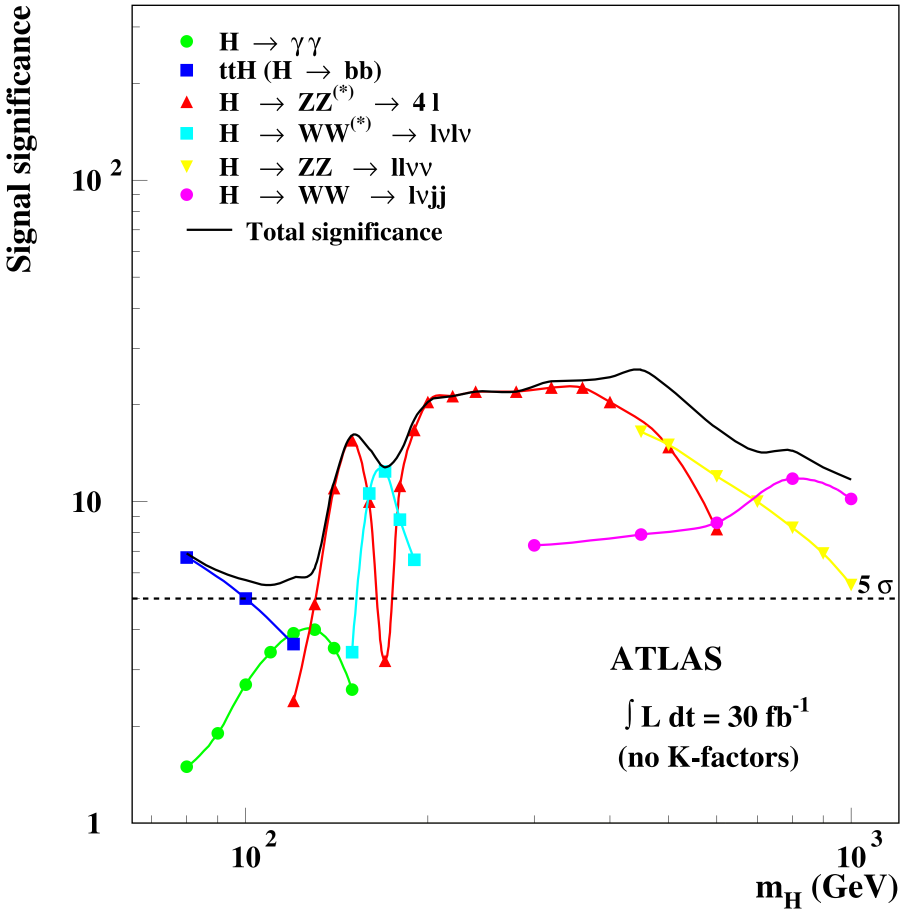

Figure 6.2: The Higgs sensitivity of ATLAS.

6.1.5 Other Searches

Other searches for Symmetry-Breaking Mechanism will be performed as well. ATLAS can be sensitive to some *Technicolor* particles, namely to technipion decaying into heavy quarks or technirho decaying into a pair of gauge bosons.

Also new heavy gauge bosons W' and Z' could be accessible to the LHC, as well as leptoguarks and composite leptons and quarks.

6.2 B-Physics

6.2.1 B-Physics in ATLAS

LHC will give enormous rate of B-hadrons thanks to the large hadronic cross-section for b-quark production. About one collision in every hundred will produce a b-quark pair, which is considerable better than at Tevatron [8]. An inclusive-muon trigger with a p_T threshold of 6 GeV will make an initial selection of B-events in ATLAS giving 25% of the muon-trigger events with a b-quark. In the first year of operation, about 10^9 events will pass the trigger. This is more than in any experiment in operation before the start-up of the LHC.

The aim of the B-physics in ATLAS is to test the Standard Model through precision measurements of B-hadron decays that will together over-constrain the *Cabibbo-Kobayashi-Maskawa CKM* matrix and may also give an indirect evidence for a new physics. CKM matrix stem from the fact that the weak eigenstates (states with definite transformation properties under the gauge group $SU(2) \times U(1)$ are not in general mass eigenstates. The matrix relating these weak and mass eigenstates is the CKM matrix. The CKM matrix can be written as follows:

$$V = \begin{pmatrix} V_{ud} & V_{us} & V_{ub} \\ V_{cd} & V_{cs} & V_{cb} \\ V_{td} & V_{ts} & V_{tb} \end{pmatrix} \tag{6.1}$$

Its 3×3 unitarity matrix $V^{\dagger}V = VV^{\dagger} = 1$ depends on three real coefficients and a phase. Those are the parameters being measured.

The full B-Physics program of ATLAS contains:

- Precise measurement of CP violation in B-meson decays.
- Precise measurement of flavor oscillation in B_s^0 and B_d^0 and relative decay rates.
- Searches for rare decay rates suppressed under the standard model.
- Precision measurements of B hadrons, including the polarization of Λ_b.

Many of those studies can be more precise in ATLAS than in low-energy experiments thanks to the larger available statistics.

Simulation

Simulations of B-physics events are divided into particle level simulations, simulations of detector response, fast simulations and dedicated simulations for some of the B-hadron decays.

About 1.8 million events were simulated with PYTHIA 5.7 and JETSET 7.4 [59]. This step, the so-called particle level simulation (PL), contains the whole chain of processes, starting with single proton-proton collision and ending with hadron decays, and also including weak decays of beauty, charm and neutral strange hyperons. All events written on the output tapes satisfy the first level trigger condition, that is, the presence of at least one muon with $p_T > 6$ GeV and $|\eta| < 2.4$. About 25% of the particle level events were processed with the detector simulation program DICE [66] based on Geant3. Background events which are rejected by extremely large factors in the analyses were simulated using a fast Monte Carlo simulation program. This includes a parametrization of the detector performance which was tuned to reproduce the results from the reconstruction of fully simulated events in the inner detector. Multiple proton-proton collisions originating from the same bunch crossing were simulated by combining hits from the inner detector simulation of one beauty event and of several unbiased events.

6.3 CP Violation Dilution Factor

For statistics 24000 of simulated events $\mathrm{B_d^0}$ B $\rightarrow$ J/ψ $\mathrm{K^0}$ μ X the rates dN/dt of reconstructed events were calculated for several options of the detector setup (air-core toroid vs. iron-core toroid, electron vs. muon tracking, full vs. staged detector) as a function of the tagging muon p_t . The dependency of a CP violation dilution factor on muon p_T arrising from misstagging of the muon was derived. From dN/dt and D the p_t -dependence of the error on CP-violation parameter $\sin(2\phi)$ was obtained. The results have been used in the ATLAS detector design.

6.3.1 Event generation

pp $\rightarrow$ b $\overline{\mathrm{b}}$ X events were generated by Pythia with only lowest-order graphs. The cross section given by Pythia for this process is 150 μb. In order to enrich statistics at higher p_t four samples of pp $\rightarrow$ b $\overline{\mathrm{b}}$ X events were generated with p_t cuts on hard-process choosen at (p_t < 6 GeV), (6 GeV < p_t <12 GeV), (12 GeV < p_t < 20 GeV), (p_t > 20 GeV). To obtain the correct spectrum the sum of four p_t - distributions of b-hadrons each weighted by a coefficient was fitted to the spectrum of b-hadrons obtained by the generation without p_t cuts.

$\mathrm{B_d^0}$ B X events were taken and forced to decay the following way

$$\mathrm{B_d^0} \rightarrow \mathrm{J}/\psi\ \mathrm{K^0} \tag{6.2}$$

and for b quark in beauty hadron B only two semileptonic decay channels were opened:

$$\mathrm{b} \rightarrow \mu^- \mathrm{c}\, \overline{\nu}_\mu \quad or\ \mathrm{b} \rightarrow \tau^- \overline{\mathrm{c}}\, \overline{\nu}_\tau \ ;\ \mathrm{b} \rightarrow \tau^- \overline{\mathrm{c}}\, \overline{\nu}_\tau \tag{6.3}$$

Four other analogical samples of events were generated with decay channels:

$$\mathrm{b} \rightarrow \mathrm{c}\ X\ ;\ \mathrm{c} \rightarrow \mu^+ X' \tag{6.4}$$

instead of (6.3); here X doesn't contain muon. The two sets of events are called $good - tag$ (with decays (6.2) and (6.3)) and $bad - tag$ (with decays (6.2) and (6.4)) respectively. Mixing of the neutral B mesons was omitted.

6.3.2 Acceptance calculations

The probability ϵ_{rec} of the

$$\mathrm{B_d^0}\ \mathrm{B} \rightarrow \mathrm{J}/\psi\ \mathrm{K^0}\ \mu\ X\ ;\ \mathrm{J}/\psi \rightarrow \mathrm{l^+l^-}\ ;\ \mathrm{K^0} \rightarrow \pi^+\pi^- \tag{6.5}$$

events (that have already satisfied trigger conditions -first two lines in Table 6.1) to fulfil the sets of cuts presented in Table 6.1 was calculated. The rapidity cuts 1.2 (0.7) for the tagging muon and 1.4 for l^+l^- and $\pi^+\pi^-$ are values taken for the staging the detector. The cuts were taken from ([93]). The lepton identification efficiency (80%) and the track-finding efficiency (95%) were taken into account.

p_t $(\mu\ tag) > 6GeV; 12GeV$	air-core; iron-core; erence
$\|\eta\|(\mu\ tag) < 2.0; 1.2; 0.7$	full detector; staged air-core; staged iron-core
p_t $(\pi^{\pm}) > 1GeV$	
$\|\eta\|(l^+l^-\ \pi^+\pi^-) < 2.0; 1.4$	full detector; staged detector
$r_{\perp}(K^0) \in < 1; 30 > cm$	
$\angle(J/\psi\ ; K^0) < 45^o$	
p_t $(l^+l^-) > 1GeV; 5GeV$	electrons; muons

Table 6.1: Cuts used for the ϵ_{rec} calculation

Number of reconstructed events (6.5) per year at luminosity 10^{33} $cm^{-2}s^{-1}$ dN/dt for different p_t triggers (Table 6.2 and also Fig. 6.4b with the cut 1 GeV for l^+,l^-) was obtained from ϵ_{rec} and N_{trig}. N_{trig} is a number of events (6.5) per year collected with a single muon trigger with $|\eta| < 2$ of tagging muon and different values of p_t trigger. N_{trig} was derived from $d\sigma/dp_t$ (Fig. 6.3a) - the differential cross section of the process (6.5) which was obtained from generated events as explained in next section.

6.3.3 Muon spectrum and dilution

The differential cross section $d\sigma/dp_t$ (Fig. 6.3a) where p_t is transversal momentum of μ from B decays was obtained as a sum of $good - tag$ and $bad - tag$ $d\sigma/dp_t$ (Fig. 6.3b) with relative normalization correspondingly to branching ratios of decays (6.3) and (6.4).

The absolute normalization of $d\sigma/dp_t$ was made in accordance to the cross section σ =0.150 mb given by PYTHIA for pp $\rightarrow$ b $\overline{\text{b}}$ X .

Asymmetry defined as:

$$A = \frac{\sigma(B_d^0\ B\ \rightarrow J/\psi\ K^0\ \mu\ ^+X\ ) - \sigma(B_d^0\ B\ \rightarrow J/\psi\ K^0\ \mu\ ^-X\ )}{\sigma(B_d^0\ B\ \rightarrow J/\psi\ K^0\ \mu\ ^+X\ ) + \sigma(B_d^0\ B\ \rightarrow J/\psi\ K^0\ \mu\ ^-X\ )} \tag{6.6}$$

measured in a pp process would differ from $\sin(2\phi) * x/(1+x^2)$, where $\sin(2\phi)$ is a CP-violation parameter and x is mixing of B_d^0 meson, for several reasons. This is usualy expressed by dilution factors. We have calculated the dilution due to misstagging t.e. when tagging muon comes from $bad-tag$ event. The ambigous events b $\rightarrow \mu^-$ c $\overline{\nu}_\mu$; c $\rightarrow \mu^+ X'$ with two muons (4.8% of events) were treated sepparately: for the certain value of trigger threshold p_t the event was considered as a $good-tag$ if $p_t(\mu^-) \geq p_t > p_t(\mu^+)$ and as a $bad-tag$ if $p_t(\mu^+) \geq p_t > p_t(\mu^-)$. We exclude also the ambigous events with both muons above the trigger threshold which will be admitted by the trigger but will have to be excluded in off-line analyses. This way for $p_t = 6$ GeV we choose 5% of the ambiguity events as $good-tag$ and 0.5% as $bad-tag$ events. The rest of 94.5% are not triggered or have ambigous tag. Two other processes leading to two tagging muon final states: B $\rightarrow$ J/ψ X; J/ψ $\rightarrow \mu^+\mu^-$ and the double pp $\rightarrow$ b $\overline{\rm b}$ X production are not considered here. From $good-tag$ and $bad-tag$ muon spectra (Fig. 6.3b) the dilution factor D defined as

$$D(p_t) = \frac{\int_{p_t}^{\infty} \frac{d\sigma(good-tag)}{dp_t{}'} dp_t{}' - \int_{p_t}^{\infty} \frac{d\sigma(bad-tag)'}{dp_t} dp_t{}'}{\int_{p_t}^{\infty} \frac{d\sigma(good-tag)}{dp_t{}'} dp_t{}' + \int_{p_t}^{\infty} \frac{d\sigma(bad-tag)'}{dp_t} dp_t{}'} \tag{6.7}$$

was calculated as a function of p_t (Fig. 6.4a).

From $D(p_t)$ and dN/dt (Fig. 6.4b) the statistical error on CP-violation parameter $\sin(2\phi)$ which is equal to

$$(x/(1+x^2) * d * D(p_t) * \sqrt{dN/dt})^{-1} \tag{6.8}$$

can be derived as a function of p_t (Fig. 6.4c). The factor d is a correction to dilution due to mixing, when all B hadrons - not only charged mesons are used for tagging. Factor d was calculated using the formula

$$d = p_- + p_\Lambda + (x + x_s)/(1 + x_s^2)/x * p_s + 1/(1 + x^2) * p_d \tag{6.9}$$

taken from [95], where x_s is a mixing parameter of $\mathrm{B_s^0}$ meson, p_-, p_d, p_s and p_Λ are the production rates of the $\mathrm{B^-}$,$\mathrm{\overline{B}_d^0}$, $\mathrm{\overline{B}_s^0}$ and the beauty baryon respectively. For our generated events we obtained the relative values of

$$p_- : p_d : p_s : p_\Lambda = (0.400 \pm 0.005) : (0.398 \pm 0.005) : (0.117 \pm 0.003) : (0.085 \pm 0.002) \tag{6.10}$$

The differences in $B/\overline{B}$ production rates in pp interactions are omitted. Using the value $x = 0.67$ [94] and $x_s = 8$ [95] the factor $d = 0.79$ is derived.

6.3.4 Results

For trigger threshold $p_t = 6$ GeV and more the dilution due to tagging is independent on p_t and is equal to

$$D = 0.82 \pm 0.04 \tag{6.11}$$

$p_t{}^{min}(\mu - tag)$ $[GeV]$	$p_t{}^{min}(l^+l^-)$ $[GeV]$	N_{trig}	ϵ_{rec} $[\%]$	dN/dt	D	$\delta(\sin(2\phi))$
6	1	37200	11.3 ± 0.8	2430	0.82 ± 0.04	0.07
(air-core)	5		2.2 ± 0.3	491		0.15
12	1	3900	17.6 ± 2.4	398	0.86 ± 0.06	0.16
(iron-core)	5		6.7 ± 1.4	151		0.26
6	1	23700	9.7 ± 0.8	1340	0.82 ± 0.04	0.09
(staged air-core)			1.9 ± 0.3	261		0.21
12	1	1930	14.0 ± 3.3	153	0.86 ± 0.06	0.26
(staged iron-core)			3.3 ± 1.2	37		0.53

Table 6.2: Results obtained for various detector setup (the air-core, the iron-core), different lepton identification limits (1 or 5 GeV) and full and staged detector.

the error on D is statistical (here the error on number of generated events is undertood) and here we take it for 6 GeV. As a consequence the error (6.8) on $\sin(2\phi)$ depends on p_t only through rates dN/dp_t . Staging would degrade this error 1.3 times for the air-core toroid or 1.6 times for the iron-core toroid.

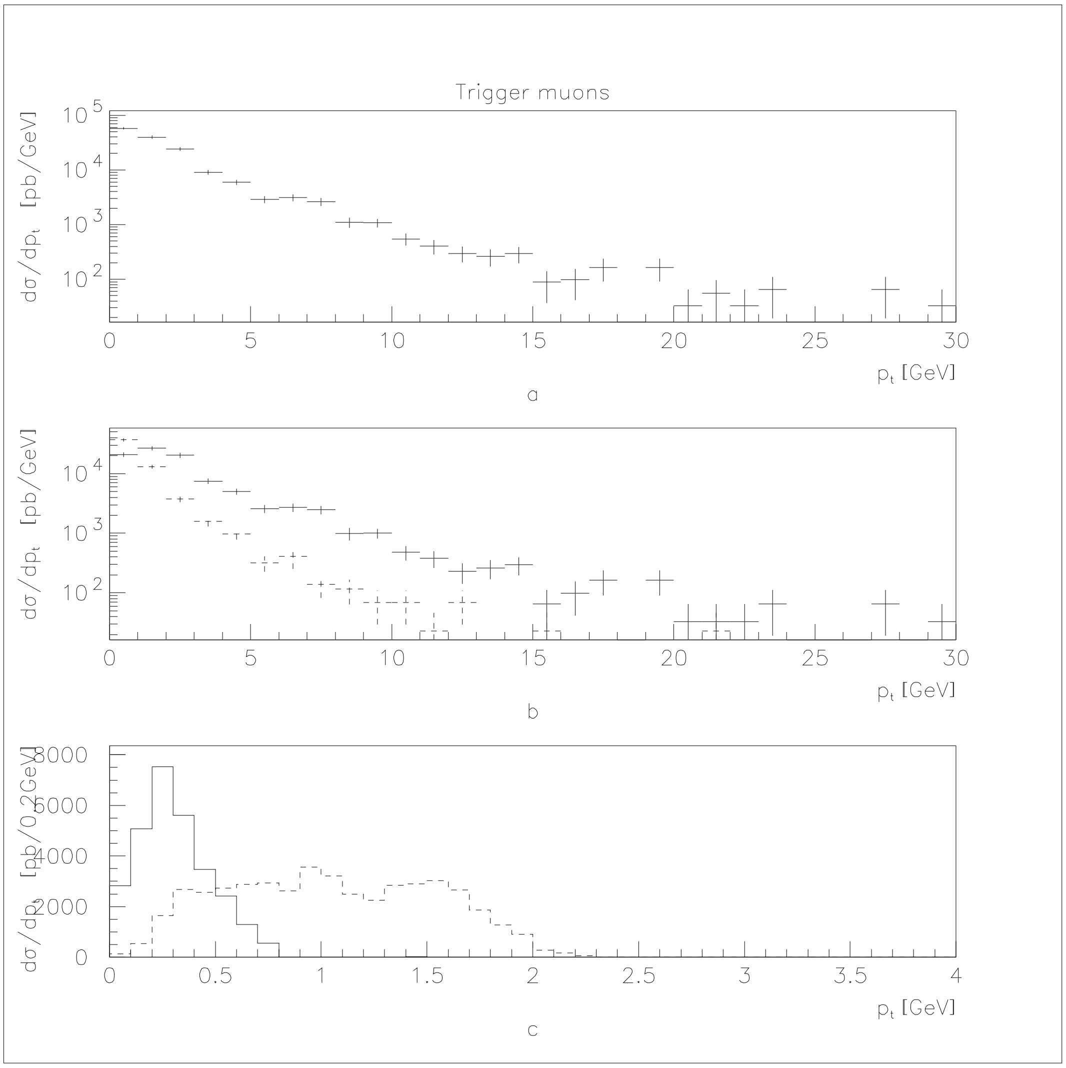

Figure 6.3: Fig.1a) p_t -distribution of muons from semileptonic B decays Fig.1b) p_t -spectra of $good-tag$ (upper points) and $bad-tag$ (lower points) muons from semileptonic B decays (The errors are statistical.) Fig.1c) p_t -spectra of $good-tag$ (dashed line) and $bad-tag$ tagging (full line) muons with respect to B direction.

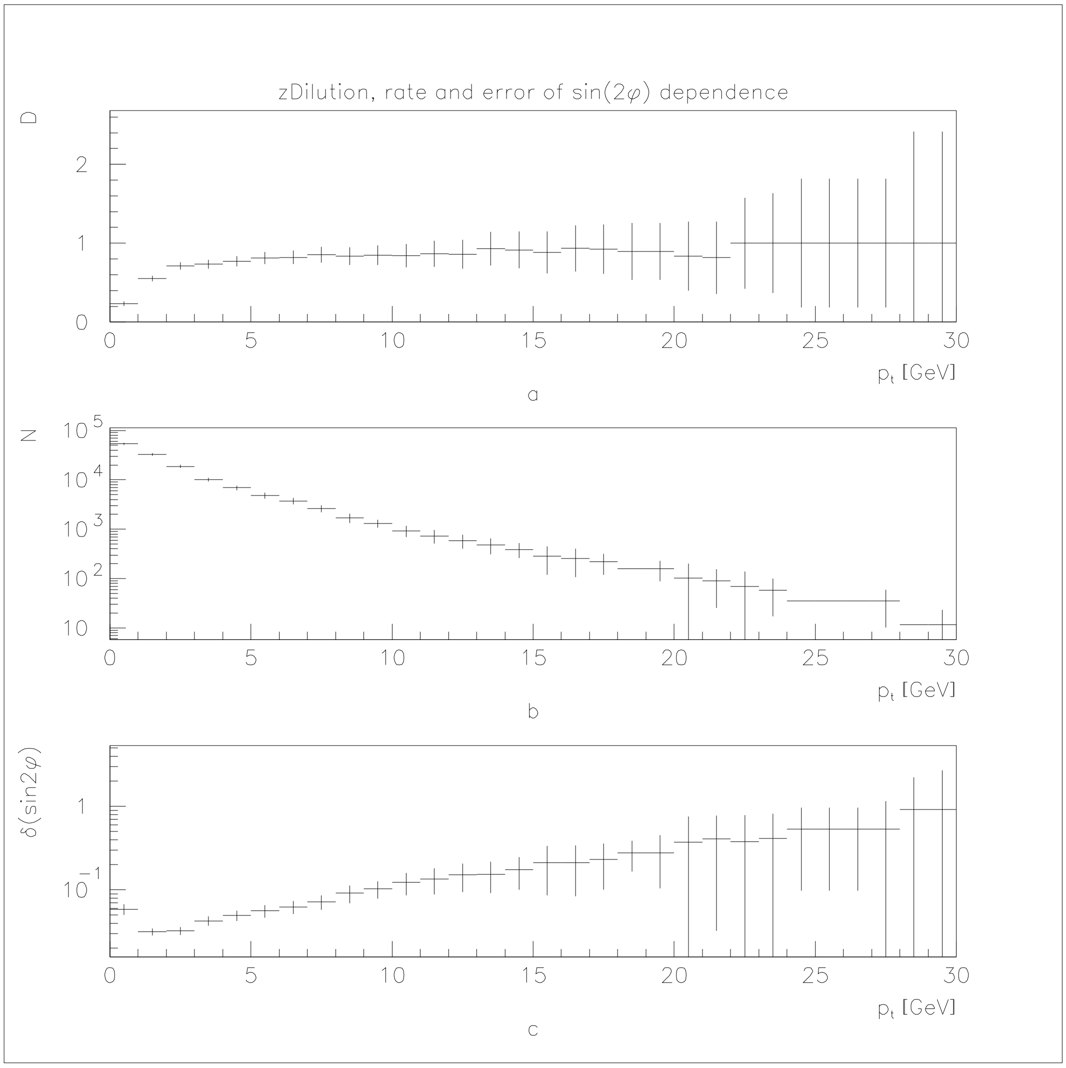

Figure 6.4: Fig.2a) Dilution factor due to muon tagging as a function of muon p_t . Fig.2b) Rates of reconstructed events J/ψ K^0 μ X ; J/ψ $\rightarrow$ l$^+$l$^-$; K^0 $\rightarrow \pi^+\pi^-$ per year at luminosity 10^{33} $cm^{-2}s^{-1}$ as a function of p_t -trigger threshold. Fig.2c) The error on CP-violance factor $sin(2\phi)$ after one year at luminosity 10^{33} $cm^{-2}s^{-1}$ as a function of p_t -trigger threshold. (The errors are statistical.)

6.4 Beauty Baryon Polarisation

The possibility of beauty baryon polarization measurement by cascade decay angular distribution analysis in the channel $\Lambda^0\ J/\psi\ \to p\pi^- l^+ l^-$ is demonstrated. The error analysis shows that in the proposed LHC experiment ATLAS at the luminosity 10^4 pb^{-1} the polarization can be measured with the statistical precision better than $\delta = 0.016$ for Λ_b^0 and $\delta = 0.28$ for Ξ_b^0 .

6.4.1 Motivation

The study of polarization effects in multiparticle production provides an important information on spin-dependence of the quark confinement. Thus substantial polarization of the hyperons produced in nucleon fragmentation processes [96, 97] as well as the data on the hadron polarization asymmetry were qualitatively described by recombination quark models taking into account the leading effect due to the valence hadron constituents [3 − 6]. Although these models correctly predict practically zero polarization of $\overline{p}$,$\overline{\Lambda}$ and Ω^-, they fail to explain the large polarization of antihyperons $\overline{\Xi^+}$ and $\overline{\Sigma^-}$ recently discovered in Fermilab [102, 103].

The problem of quark polarization effects could be clarified in polarization measurements involving heavy quarks. In particular, an information about the quark mass dependence of these effects could be obtained [99, 104]. The polarization is expected to be proportional to the quark mass if it arises due to scattering on a colour charge [10 − 12]. The opposite dependence takes place if the quark becomes polarized due to the interaction with an "external" confining field, e.g., due to the effect of spontaneous radiation polarization [108]. The decrease of the polarization with increasing quark mass is expected also in the model of ref. [109].

In QCD the polarization might be expected to vanish with the quark mass due to vector character of the quark-gluon coupling [105]. It was shown however in Ref.[110] that the quark mass should be effectively replaced by the hadron mass M so that even the polarization of ordinary hadrons can be large. The polarization is predicted to be independent of energy and to vanish in the limit of both low and high hadron transverse momentum p_t . The maximal polarization $P_{max}(x_F)$ is reached at $p_t\ \approx M$ and depends on the Feynman variable x_F. Its magnitude (and in particular its mass dependence) is determined by two quark-gluon correlators which are not predicted by perturbative QCD.

The polarization of charm baryons in hadronic reactions is still unmeasured due to the lack of sufficient statistics. Only some indications on a nonzero polarization were reported [111, 112]. For beauty physics the future experiments on LHC or HERA give an opportunity to obtain large statistical samples of beauty baryon (Λ_b^0 , Ξ_b^0) decays into Λ^0 J/ψ $\to p\pi^- l^+ l^-$, which is favorable mode to detect experimentally. Dedicated triggers for CP-violation effects in b-decays, like the high-p_t one-muon trigger (LHC) [113] or the J/ψ trigger (HERA) [114] are selective also for this channel.

Below we consider the possibility of polarization measurement of beauty baryons Λ_b^0 and Ξ_b^0 with the help of cascade decay angular distributions in the channel Λ_b^0 (Ξ_b^0) $\to \Lambda^0$ J/ψ $\to p\pi^- l^+ l^-$.

6.4.2 Polarization measurement method and an estimation of the statistical error.

In the case of parity nonconserving beauty baryon (B_b) decay the polarization causes the asymmetry of the distribution of the cosine of the angel θ between the beauty baryon decay and production analyzers:

$$w(\cos\theta) = \frac{1}{2}(1 + \alpha_b P_b \cos\theta), \tag{6.12}$$

where P_b is a projection of the polarization vector on the production analyzer and α_b is a decay asymmetry parameter. As the polarization vector is perpendicular to B_b production plane (due to parity conservation in the production process), the production analyzer should be directed parallel to the normal to this plane: $\vec{n} = \frac{\vec{p}_{inc} \times \vec{p}_{B_b}}{|\vec{p}_{inc} \times \vec{p}_{B_b}|}$, where $\vec{p}_{inc}$ and $\vec{p}_{B_b}$ are momenta of incident particle and B_b in c.m. system.

The asymmetry parameter α_b characterizes parity nonconservation in a weak decay of B_b and depends on the choice of the decay analyzer. In the two-body decay $B_b \to \Lambda^0$ J/ψ it is natural to choose this analyzer oriented in the direction of Λ^0 momentum $\vec{p}_{\Lambda^0}$ in the B_b rest system. The considered decay can be described by 4 helicity amplitudes $A(\lambda_1, \lambda_2)$ normalized to unity: $a_+ = A(1/2, 0)$,$a_- = A(-1/2, 0)$, $b_+ = A(-1/2, 1)$ and $b_- = A(1/2, -1)$,

$$|a_+|^2 + |a_-|^2 + |b_+|^2 + |b_-|^2 = 1. \tag{6.13}$$

The difference of Λ^0 and J/ψ helicities λ_1 - λ_2 is just a projection of B_b spin onto the decay analyzer. The decay asymmetry parameter α_b is expressed through these amplitudes in the form

$$\alpha_b = |a_+|^2 - |a_-|^2 + |b_+|^2 - |b_-|^2. \tag{6.14}$$

If P-parity in B_b decay were conserved, then $|a_+|^2 = |a_-|^2$, $|b_+|^2 = |b_-|^2$ so that α_b would be 0. In the case of known and sufficiently nonzero value of

α_b the beauty baryon polarization could be simply measured with the help of angular distribution (6.12) (see, e.g., [115]). Due to lack of experimental information and rather uncertain theoretical estimates of α_b for the decay $\Lambda_b^0 \rightarrow \Lambda^0\ J/\psi$ [116] both the polarization and α_b (or the decay amplitudes) should be determined simultaneously. This can be achieved with the help of information on Λ^0 and J/ψ decays. Though it complicates the analysis, it should be stressed that the measurement of the beauty baryon decay amplitudes could give valuable constrains on various theoretical models. Generally, such a measurement can be done provided that at least one of the secondary decays is asymmetric and its decay asymmetry parameter is known [104]. In our case it is the decay $\Lambda^0 \rightarrow p\pi^-$ with the asymmetry parameter α_Λ=0.642.

The angular distribution in the cascade decay $B_b \rightarrow \Lambda^0\ J/\psi \rightarrow p\pi^- l^+ l^-$ follows directly from Eq. (6) of [104], taking into account that the only nonzero multipole parameters in the decay $J/\psi \rightarrow l^+ l^-$ are T_{00}=1 and $T_{20}=\frac{1}{\sqrt{10}}$. It can be written in the form

$$w(\Omega, \Omega_1, \Omega_2) = \frac{1}{(4\pi)^3} \sum_{i=0}^{i=19} f_{1i} f_{2i}(P_b, \alpha_\Lambda) F_i(\theta, \theta_1, \theta_2, \phi_1, \phi_2), \tag{6.15}$$

where f_{1i} are bilinear combinations of the decay amplitudes a_+, a_-, b_+, b_-, $f_{2i}(P_b, \alpha_\Lambda)$ stands for $P_b\alpha_\Lambda$, P_b, α_Λ or 1 and F_i are orthogonal angular functions (Table6.3). Ω=(θ,ϕ) are the polar and the azimuthal angles of the Λ^0 momentum $\vec{p}_{\Lambda^0}$ in the B_b rest frame with z-axis parallel to the production normal $\vec{n}$ (the choice of x and y axes is not important since parity conservation in the production process guarantees independence of the azimuthal angle ϕ), Ω_1=(θ_1,ϕ_1) are the angles of proton momentum in the Λ^0 rest frame with axes defined as $z_1 \uparrow\uparrow \vec{p}_{\Lambda^0}$, $y_1 \uparrow\uparrow \vec{n} \times \vec{p}_{\Lambda^0}$, Ω_2=(θ_2,ϕ_2) are the angles of the momentum of one of the decay leptons in the J/ψ rest frame with axes defined as $z_2 \uparrow\uparrow \vec{p}_{J/\psi}$, $y_2 \uparrow\uparrow \vec{n} \times \vec{p}_{J/\psi}$ ($\vec{p}_{J/\psi}$=$-\vec{p}_{\Lambda^0}$ is J/ψ momentum in the B_b rest frame). See Fig. 6.5.

This distribution depends on 7 unknown independent parameters. One of them is the polarization P_b and six others determine the four complex amplitudes

$$a_+ = |a_+|e^{i\alpha_+}, a_- = |a_-|e^{i\alpha_-}, b_+ = |b_+|e^{i\beta_+}, b_- = |b_-|e^{i\beta_-}. \tag{6.16}$$

Taking into account the normalization condition and arbitrariness of the common phase, these parameters can be chosen as

$$\alpha_b, r_0 = |a_+|^2 + |a_-|^2, |a_+|^2 - |a_-|^2, \alpha_+, \alpha_-, \chi = \alpha_+ + \alpha_- - \beta_+ - \beta_-. \tag{6.17}$$

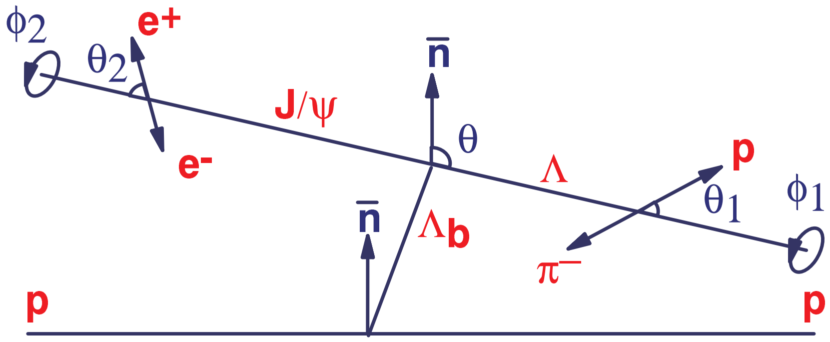

Figure 6.5: The definition of the five angles. θ is the polar angle of the Λ^0 momentum $\vec{p}_{\Lambda^0}$ in the Λ_b^0 rest frame relative to the normal $\vec{n}$ to the production plane. θ_1 and ϕ_1 are the angles of the proton in Λ^0 rest frame, with the z-axis being parallel $p_{\vec{\Lambda}^0}$ and the y-axis being parallel to $\vec{n} \times p_{\vec{\Lambda}^0}$. θ_2 and ϕ_2 are defined for the J/ψ decay in a similar way.

The parameters can be determined from the measured decay angles by a 5-dimensional likelihood fit or by the moment method: the angular distribution of the form (6.15) allows one to introduce 19 moments $< F_i >\sim f_{1i}.f_{2i}$ and determine the parameters from their measured values by weighted least-squares method. For the polarization measurement the formula (6.15) integrated over the azimuthal angles ϕ_1, ϕ_2 would be in principle sufficient [104]. In this case the number of free parameters is reduced to 4 (the phases don't enter) and only a 3-dimensional fit is required. We will see, however, that the information on these angles may substantially increase the precision of the P_b determination.

To simplify the error analysis, we follow ref. [104] and consider here only the most unfavourable situation, when the parameters P_b^2,$|a_+|^2 - |a_-|^2$ and $|b_+|^2 - |b_-|^2$ are much smaller than α_Λ^2. In this case the moments $< F_i >$ can be considered to be independent, having the diagonal error matrix

$$W = \frac{1}{N} diag(\frac{1}{3}, \frac{1}{3}, \frac{1}{9}, \frac{1}{5}, \frac{1}{15}, \frac{1}{15}, \frac{1}{45}, \frac{16}{135}, \frac{16}{135}, \frac{16}{135}, \frac{16}{135}, \frac{2}{135}, \frac{2}{135}, \frac{2}{135}, \frac{2}{135}, \frac{2}{45}, \frac{2}{45}, \frac{2}{45}, \frac{2}{45}). \tag{6.18}$$

Here N is a number of B_b events (assuming that the background can be neglected, see next section). The error matrix V of the vector $\vec{a}$ of the parameters a_j,j=1,..7 defined in (6.17) is

$$V(\vec{a}) = (A^T W^{-1} A)^{-1}, \tag{6.19}$$

where the elements of the matrix A are $A_{ij} = \frac{d(f_{1i}.f_{2i})}{da_j}$. In the considered situation the error on the polarization P_b is

$$\delta \equiv \sqrt{V_{11}} = \frac{\delta_0}{\sqrt{N}}, \tag{6.20}$$

$$\delta_0 = \frac{1}{\sqrt{\alpha_\Lambda^2 \cdot [\frac{(2r_0-1)^2}{9} + \frac{(r_0+1)^2}{180} + \frac{4r_0^2}{15} + \frac{(1-r_0)^2}{15} + \frac{(1-r_0)(1+\cosh\chi)}{15}] + \frac{r_0(1-r_0)(1-\cosh\chi)}{10}}}. \tag{6.21}$$

Here δ_0 depends only on the relative contribution r_0 of the decay amplitudes with helicity $\lambda_2 = 0$ and on the relative phase χ (Figs. 6.6a,b). The maximal error on P_b is $\delta_{\max} = \frac{4.7}{\sqrt{N}}$ and it corresponds to the case when $r_0 = \frac{1}{3}$ and the phase $\chi = 0$. The minimal error in the considered most unfavorable situation is $\delta_{\min} = \frac{2.5}{\sqrt{N}}$. It is obtained in the case when only $\lambda_2 = 0$ helicity amplitudes contribute ($r_0 = 1$), independently of their phases. If the information on the azimuthal angles would be neglected then $\delta_{\max} = \frac{14.0}{\sqrt{N}}$ and $\delta_{\min} = \frac{4.3}{\sqrt{N}}$.

6.4.3 Estimates of indirect Λ_b^0 and Ξ_b^0 production, background processes and possible statistics at ATLAS-LHC

The polarization transfered to Λ_b^0 , which was produced indirectly in strong Σ_b and Σ_b^* decays, depends on the ratio $\frac{\Delta}{\Gamma}$ of the mass difference Δ between Σ_b and Σ_b^* and the decay rate Γ of $\Sigma_b \to \Lambda_b^0\,\pi$ or $\Sigma_b^* \to \Lambda_b^0\,\pi$ [120]. We will consider the limit $\Delta >> \Gamma$ when the baryons Σ_b and Σ_b^* form two well separated resonances and their contributions to Λ_b^0 polarization can be added incoherently. In such a case the polarization transfer can be easily calculated in a model-independent way using the angular distributions in cascade decays [121]. Thus considering the cascade decay $\Sigma_b \to \Lambda_b^0\,\pi$ or $\Sigma_b^* \to \Lambda_b^0\,\pi$, $\Lambda_b^0 \to \Lambda^0\,J/\psi$, we have

$$w(\Omega,\Omega_1) \sim 1 + \alpha_{\Lambda_b^0} P_{\Sigma_b}(\cos\theta\cos\theta_1 \pm \sin\theta\sin\theta_1\cos\phi_1) \tag{6.22}$$

for Σ_b (Σ_b^*). Here Ω=(θ,ϕ) are the polar and the azimuthal angles of the Λ_b^0 momentum $\vec{p}_{\Lambda_b^0}$ in the $\Sigma_b(\Sigma_b^*)$ rest frame with z-axis parallel to the production normal: $\vec{n} = \frac{\vec{p}_{inc}\times\vec{p}_{\Sigma_b}}{|\vec{p}_{inc}\times\vec{p}_{\Sigma_b}|}$, where $\vec{p}_{inc}$ and $\vec{p}_{\Sigma_b}$ are momenta of incident particle and Σ_b in c.m. system. Ω_1=(θ_1,ϕ_1) are the polar and the azimuthal angles of Λ^0 in Λ_b^0 rest frame with the axes defined as $z_1 \uparrow\uparrow \vec{p}_{\Lambda_b^0}$, $y_1 \uparrow\uparrow \vec{n}\times\vec{p}_{\Lambda_b^0}$. After the transformation of $\Omega_1 \to \Omega_1'$ of Λ^0 angles from the helicity frame x_1, y_1, z_1 to the canonical frame x, y, z with $z \uparrow\uparrow \vec{n}$ and the integration over $\cos\theta$ and $\phi_1^{'}$ we get the distribution of the cosine of the angle between the Λ^0 momentum vector (Λ_b^0 decay analyzer) and the $\Sigma_b or \Sigma_b^*$ production normal (which can be considered coinciding with the Λ_b^0 production normal due to a small energy release in the $\Sigma_b or \Sigma_b^*$ decays):

$$w(\cos\theta_1^{'}) \sim 1 \mp \frac{1}{3}\alpha_{\Lambda_b^0} P_{\Sigma_b} \cos\theta_1^{'} \tag{6.23}$$

for Σ_b (Σ_b^*) decay. One can thus conclude that Λ_b^0 retains $\frac{-1}{3}(\frac{1}{3})$ of the Σ_b (Σ_b^*) polarization. The analogical analyses of electromagnetic decays $\Xi_b^{0'} \to \Xi_b^0\,\gamma$ or $\Xi_b^{0*} \to \Xi_b^0\,\gamma$ show that Ξ_b^0 retains $\frac{-1}{3}(\frac{1}{3})$ of the $\Xi_b^{0'}(\Xi_b^{0*})$ polarization.

The beauty baryons Λ_b^0 and Ξ_b^0 are produced directly or through the decays of heavier states. According to the PYTHIA tables we consider here the strong decays $\Sigma_b \to \Lambda_b^0\,\pi$, $\Sigma_b^* \to \Lambda_b^0\,\pi$ and the electromagnetic decays $\Xi_b^{0'} \to \Xi_b^0\,\gamma$ or $\Xi_b^{0*} \to \Xi_b^0\,\gamma$.The observable polarization P_{obs} depends on the polarizations P_{B_b} of direct beauty baryons and their production fractions b_{B_b} (i.e. probabilities of the b-quark to hadronize to certain baryons B_b). As has been shown above, in considered decays the beauty baryon Λ_b^0 or Ξ_b^0 retains $\frac{-1}{3}(\frac{1}{3})$ of the polarization of a parent with spin $\frac{1}{2}^+(\frac{3}{2}^+)$. For P_{obs} we get:

$$P_{obs} = \frac{b_{\Lambda_b^0} P_{\Lambda_b^0} + \sum_i(\frac{-1}{3} b_{\Sigma_{bi}} P_{\Sigma_{bi}} + \frac{1}{3} b_{\Sigma_{bi}^*} P_{\Sigma_{bi}^*})}{b_{\Lambda_b^0} + \sum_i(b_{\Sigma_{bi}} + b_{\Sigma_{bi}^*})}, \tag{6.24}$$

for Λ_b^0 and for Ξ_b^0 :

$$P_{obs} = \frac{b_{\Xi_b^0} P_{\Xi_b^0} + \frac{-1}{3} b_{\Xi_b^{0'}} P_{\Xi_b^{0'}} + \frac{1}{3} b_{\Xi_b^{0*}} P_{\Xi_b^{0*}}}{b_{\Xi_b^0} + b_{\Xi_b^{0'}} + b_{\Xi_b^{0*}}}. \tag{6.25}$$

The summation goes over positive, negative and neutral Σ_b and Σ_b^*. Assuming the polarization of the heavier states to be similar in magnitude to that of directly produced Λ_b^0 or Ξ_b^0 ($P_{\Lambda_b^0}$ or $P_{\Xi_b^0}$) we may expect the observed polarization in an interval of $(0.34 - 0.67)P_{\Lambda_b^0}$ for Λ_b^0 and $(0.69 - 0.84)P_{\Xi_b^0}$ for Ξ_b^0 .

The polarization can be measured for Λ_b^0 and Ξ_b^0 baryons and for their antiparticles. Λ_b^0 (Ξ_b^0) are unambigously distinquishable from their antiparticles by effective mass of $p\pi^-$ system from $\Lambda^0 \to p\pi^-$ decay. The wrong assignment of antiproton and pion masses gives the kinematical reflection of Λ^0 starting at 1.5 GeV (Fig. 6.7) which is far from Λ^0 mass comparing to the mass resolution of Λ^0 produced in the decays of Λ_b^0 or Ξ_b^0 .

Λ_b^0 and Ξ_b^0 can be distinguished from each other by the effective mass of the $p\pi^- l^+ l^-$ system. The mass difference of Λ_b^0 and Ξ_b^0 is 220 MeV. The mass resolution in this region (taking into account the table mass values for Λ^0 and J/ψ candidates) is $\sigma_{\Lambda^0 J/\psi} \approx 26$ MeV for Λ^0 J/ψ $\to p\pi^-\mu^+\mu^-$ channel. Thus even the background from Λ_b^0 in the region of masses $m_{\Xi_b^0} \pm 3\sigma_{\Lambda^0 J/\psi}$ is negligible.

There are several other sources of background to Λ^0 J/ψ decays of Λ_b^0 and Ξ_b^0 . The dominant background comes from J/ψ from b-hadron decays and Λ^0 produced in fragmentation or in the decay of the same b-hadron. After the cutting off the low transverse momenta (less than 0.5 GeV) of p and π from Λ^0 decay this background can be reduced to $\approx$ 1.5% in a region of $m_{\Lambda_b^0} \pm 3\sigma_{\Lambda_b^0 J/\psi}$(Figs. 6.8a,b). The remaining background comes mainly from two processes: Ξ_b^0 $\to \Xi^0 J/\psi$, $\Xi^0 \to \Lambda^0$ π^0 and $\Xi_b^- \to \Xi^- J/\psi$, $\Xi^- \to \Lambda^0$ π^- . For Ξ_b^0 this background (Fig.6.9a) is more important ($\approx$ 20%). The reason is that the decay Ξ_b^0 is governed by $b \to dc\bar{c}$ process which is Cabbibo suppressed by a factor $\sin(\theta_c)^2 = 0.22^2$ compared to the decays Ξ_b^0 $\to \Xi^0 J/\psi$, $\Xi_b^- \to \Xi^- J/\psi$ and also Λ_b^0 $\to \Lambda^0$ J/ψ , which are governed by $b \to sc\bar{c}$. However Λ^0 from Ξ_b^0 $\to \Xi^0 J/\psi$ or $\Xi_b^- \to \Xi^- J/\psi$ is produced in a weak hyperon decay, so this background can be efficiently reduced by the cut on the minimal distance d between J/ψ and Λ^0 . A conservative cut d <1.5 mm reduces this background by a factor $\approx$ 0.05 (Fig. 6.9b).

The background from $B_d^0 \to J/\psi K^0$ when one of π mesons is considered as a proton is negligible after the effective mass cuts on $(p\pi)$ and $(p\pi J/\psi)$ systems.

Background from fake $J/\psi's$, as it has been shown in [113], can be reduced to a low level by cuts on the distance between the primary vertex and the production point of the J/ψ candidate and the distance of closest approach between the two particles from the decay. These cuts also suppress the background from real $J/\psi's$ comming directly from hadronization.

The number of produced Λ_b^0 and Ξ_b^0 is calculated for the cross section of pp $\to$ b $\bar{\text{b}}$ X equal to 500 μb. The production fraction $b_{\Lambda_b^0}$ of $b \to \Lambda_b$ multiplied by branching ratio $br_{\Lambda_b^0}$ of Λ_b^0 $\to \Lambda^0$ J/ψ decay was measured by CDF[118] and is 3.7×10^{-5}. The production fraction of Ξ_b^0 given by PYTHIA is $b_{\Xi^0} = 5.5\ 10^{-3}$, where also Ξ_b^0 from electromagnetic decays of $\Xi_b^{0'}$ and Ξ_b^{0*} are taken into account. The branching ratio of the decay Ξ_b^0 $\to \Lambda^0$ J/ψ can be roughly estimated multiplying $br_{\Lambda_b^0}$ by the Cabbibo suppression factor $\sin(\theta_c)^2 = 0.22^2$. Thus the number of Λ_b^0 $\to \Lambda^0$ J/ψ processes can be expected to be $\approx$ 300 times larger than the number of Ξ_b^0 $\to \Lambda^0$ J/ψ ones.

Both channels Λ^0 J/ψ $\to p\pi^-\mu^+\mu^-$ and Λ^0 J/ψ $\to p\pi^-e^+e^-$ can be used for the analysis. However statistics will be dominated by the former decay as the trigger can be satisfied by one of the two muons. In the case of

Λ^0 $J/\psi \rightarrow p\pi^- e^+ e^-$ the trigger comes only from semileptonic decay $b \rightarrow \mu X$ of the associated b-hadron. The reconstruction efficiencies are calculated by demanding that the events satisfy the cuts listed bellow. The first set of cuts corresponds the requirements set by the trigger:

- For Λ^0 $J/\psi \rightarrow p\pi^- \mu^+ \mu^-$ decay one muon is required to satisfy the single muon trigger conditions: $p_\perp^\mu > 6$ GeV and $|\eta| < 1.6$. For the second muon the cuts $p_\perp^\mu > 3$ GeV,$|\eta| < 2.5$ are applied, supposing that within these values muon can be identified using the last segment of the hadron calorimeter by its minimum ionizing signature.

- For Λ^0 $J/\psi \rightarrow p\pi^- e^+ e^-$ decay both electrons are required to have $p_\perp^e > 1GeV$. The low threshold for electrons is possible, because of electron identification in the transition radiation tracker (TRT) [119]. The events are required to contain one muon with a $p_\perp^\mu > 6GeV$ and $|\eta| < 1.6$

The second set of cuts corresponds to 'offline' analysis cuts. The same cuts as for $B_d^0 \rightarrow J/\psi K^0$ reconstruction [113] can be used (the only exception is the mass requirement for Λ^0 candidate, see the last of the next cuts) :

- The two charged hadrons from Λ^0 decay are required to be within the tracking volume $|\eta| < 2.5$, and transverse momenta of both to be greater than 0.5 GeV.

- Λ^0 decay length in the transverse plane with respect to the beam axis was required to be greater than 1 cm and less than 50 cm. The upper limit ensures that the charged tracks from Λ^0 decay start before the inner radius of TRT, and that there is a space point from the innermost layer of the outer silicon tracker. The lower limit reduces the combinatorial background from particles originating from the primary vertex.

- The distance of closest approach between the two muon (electron) candidates forming the J/ψ was required to be less than 320 μm (450 μm), giving an acceptance for signal of 0.94.

- The proper time of the Λ_b^0 decay, measured from the distance between the primary vertex and the production point of the J/ψ in the transverse plane and the reconstructed $p_\perp$ of Λ_b^0 , was required to be greater than 0.5 ps. This cut is used to reduce the combinatorial background, giving the acceptance for signal events 0.68.

- The reconstructed Λ^0 and J/ψ masses were required to be within two standart deviations of nominal values.

For both channels the statistics of reconstructed events at the luminosity 10^4 pb^{-1} will be
75 000 Λ_b^0 and 250 Ξ_b^0 , where the values are derived using CDF results.

For this statistics the maximal value of the statistical error on the polarization measurement, calculated from formulae (6.20) and (6.21), will be 0.016 for Λ_b^0 and 0.28 for Ξ_b^0 .

6.4.4 Conclusion

At LHC luminosity 10^4 pb^{-1} the beauty baryons Λ_b^0 and Ξ_b^0 polarizations can be measured with the help of angular distributions in the cascade decays $\Lambda^0\ J/\psi\ \to p\pi^-\mu^+\mu^-$ and $\Lambda^0\ J/\psi\ \to p\pi^- e^+e^-$ with the statistical precision better than 0.010 for Λ_b^0 and 0.17 for Ξ_b^0 .

i	f_{1i}	f_{2i}	F_i
0	$a_+a_+^* + a_-a_-^* + b_+b_+^* + b_-b_-^*$	1	1
1	$a_+a_+^* - a_-a_-^* + b_+b_+^* - b_-b_-^*$	P_b	$\cos\theta$
2	$a_+a_+^* - a_-a_-^* - b_+b_+^* + b_-b_-^*$	α_Λ	$\cos\theta_1$
3	$a_+a_+^* + a_-a_-^* - b_+b_+^* - b_-b_-^*$	$P_b\alpha_\Lambda$	$\cos\theta\cos\theta_1$
4	$-a_+a_+^* - a_-a_-^* + \frac{1}{2}b_+b_+^* + \frac{1}{2}b_-b_-^*$	1	$d_{00}^2(\theta_2)$
5	$-a_+a_+^* + a_-a_-^* + \frac{1}{2}b_+b_+^* - \frac{1}{2}b_-b_-^*$	P_b	$d_{00}^2(\theta_2)\cos\theta$
6	$-a_+a_+^* + a_-a_-^* - \frac{1}{2}b_+b_+^* + \frac{1}{2}b_-b_-^*$	α_Λ	$d_{00}^2(\theta_2)\cos\theta_1$
7	$-a_+a_+^* - a_-a_-^* - \frac{1}{2}b_+b_+^* - \frac{1}{2}b_-b_-^*$	$P_b\alpha_\Lambda$	$d_{00}^2(\theta_2)\cos\theta\cos\theta_1$
8	$-3Re(a_+a_-^*)$	$P_b\alpha_\Lambda$	$\sin\theta\sin\theta_1\sin^2\theta_2\cos\phi_1$
9	$3Im(a_+a_-^*)$	$P_b\alpha_\Lambda$	$\sin\theta\sin\theta_1\sin^2\theta_2\sin\phi_1$
10	$-\frac{3}{2}Re(b_-b_+^*)$	$P_b\alpha_\Lambda$	$\sin\theta\sin\theta_1\sin^2\theta_2 cos(\phi_1+2\phi_2)$
11	$\frac{3}{2}Im(b_-b_+^*)$	$P_b\alpha_\Lambda$	$\sin\theta\sin\theta_1\sin^2\theta_2 sin(\phi_1+2\phi_2)$
12	$-\frac{3}{\sqrt{2}}Re(b_-a_+^* + a_-b_+^*)$	$P_b\alpha_\Lambda$	$\sin\theta\cos\theta_1\sin\theta_2\cos\theta_2\cos\phi_2$
13	$\frac{3}{\sqrt{2}}Im(b_-a_+^* + a_-b_+^*)$	$P_b\alpha_\Lambda$	$\sin\theta\cos\theta_1\sin\theta_2\cos\theta_2\sin\phi_2$
14	$-\frac{3}{\sqrt{2}}Re(b_-a_-^* + a_+b_+^*)$	$P_b\alpha_\Lambda$	$\cos\theta\sin\theta_1\sin\theta_2\cos\theta_2\cos(\phi_1+\phi_2)$
15	$\frac{3}{\sqrt{2}}Im(b_-a_-^* + a_+b_+^*)$	$P_b\alpha_\Lambda$	$\cos\theta\sin\theta_1\sin\theta_2\cos\theta_2\sin(\phi_1+\phi_2)$
16	$\frac{3}{\sqrt{2}}Re(a_-b_+^* - b_-a_+^*)$	P_b	$\sin\theta\sin\theta_2\cos\theta_2\cos\phi_2$
17	$-\frac{3}{\sqrt{2}}Im(a_-b_+^* - b_-a_+^*)$	P_b	$\sin\theta\sin\theta_2\cos\theta_2\sin\phi_2$
18	$\frac{3}{\sqrt{2}}Re(b_-a_-^* - a_+b_+^*)$	α_Λ	$\sin\theta_1\sin\theta_2\cos\theta_2\cos(\phi_1+\phi_2)$
19	$-\frac{3}{\sqrt{2}}Im(b_-a_-^* - a_+b_+^*)$	α_Λ	$\sin\theta_1\sin\theta_2\cos\theta_2\sin(\phi_1+\phi_2)$

Table 6.3: The coefficients f_{1i}, f_{2i} and angular functions F_i in distribution (6.15).

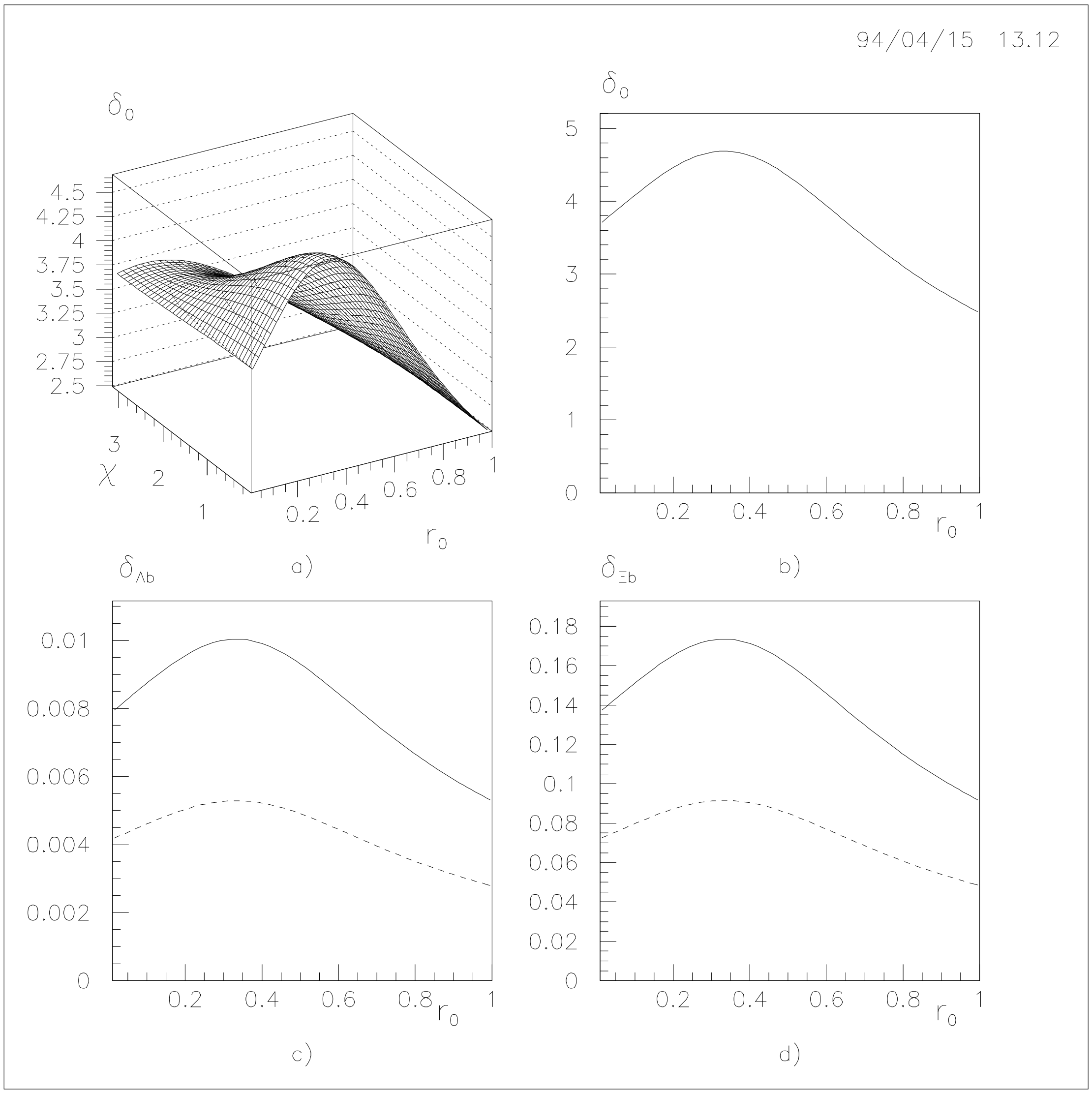

Figure 6.6: The maximal statistical error on the polarization measurement $\delta(P_b)$ and $\delta_0 = N^{+\frac{1}{2}}.\delta(P_b)$ defined in (6.21): δ_0 as a function of the relative contribution r_0 of the decay amplitudes with zero J/ψ helicity and of the relative phase χ of the amplitudes with J/ψ helicity equal to 0 and $\pm 1(a)$. The $\chi = 0$ projection of this function (b) . Dependence of $\delta(P_b)$ on r_0 at $\chi = 0$ for Λ_b^0 expected on ATLAS at LHC luminosity $10^4 pb^{-1}$ (c). Full (dotted) line corresponds to $\delta(P_b)$ derived from CDF (UA1) data. The same as (c) for Ξ_b^0 (d).

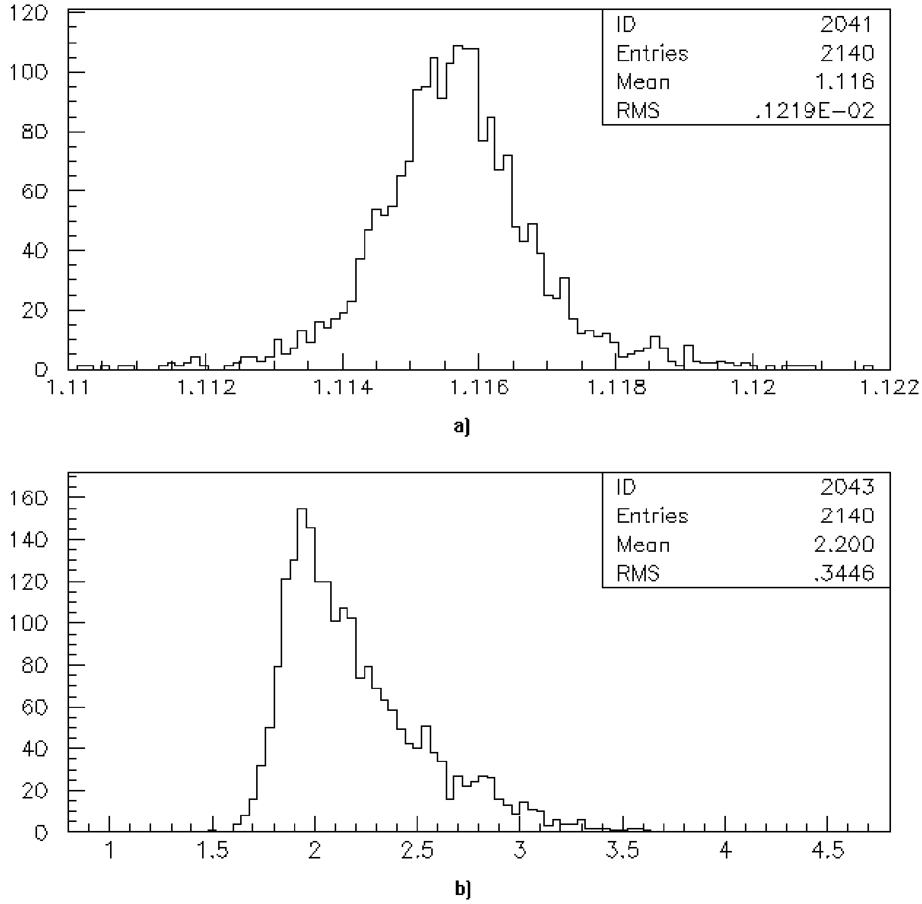

Figure 6.7: Λ^0 /$\overline{\Lambda^0}$ separation by effective mass spectrum of $p\pi^-$. Effective mass spectrum of Λ^0 from $\Xi_b^0 \rightarrow \Lambda^0 \; J/\psi$ (a). Effective mass spectrum for the same V_0's with proton mass assigned to negative track and π mass to positive track (b).

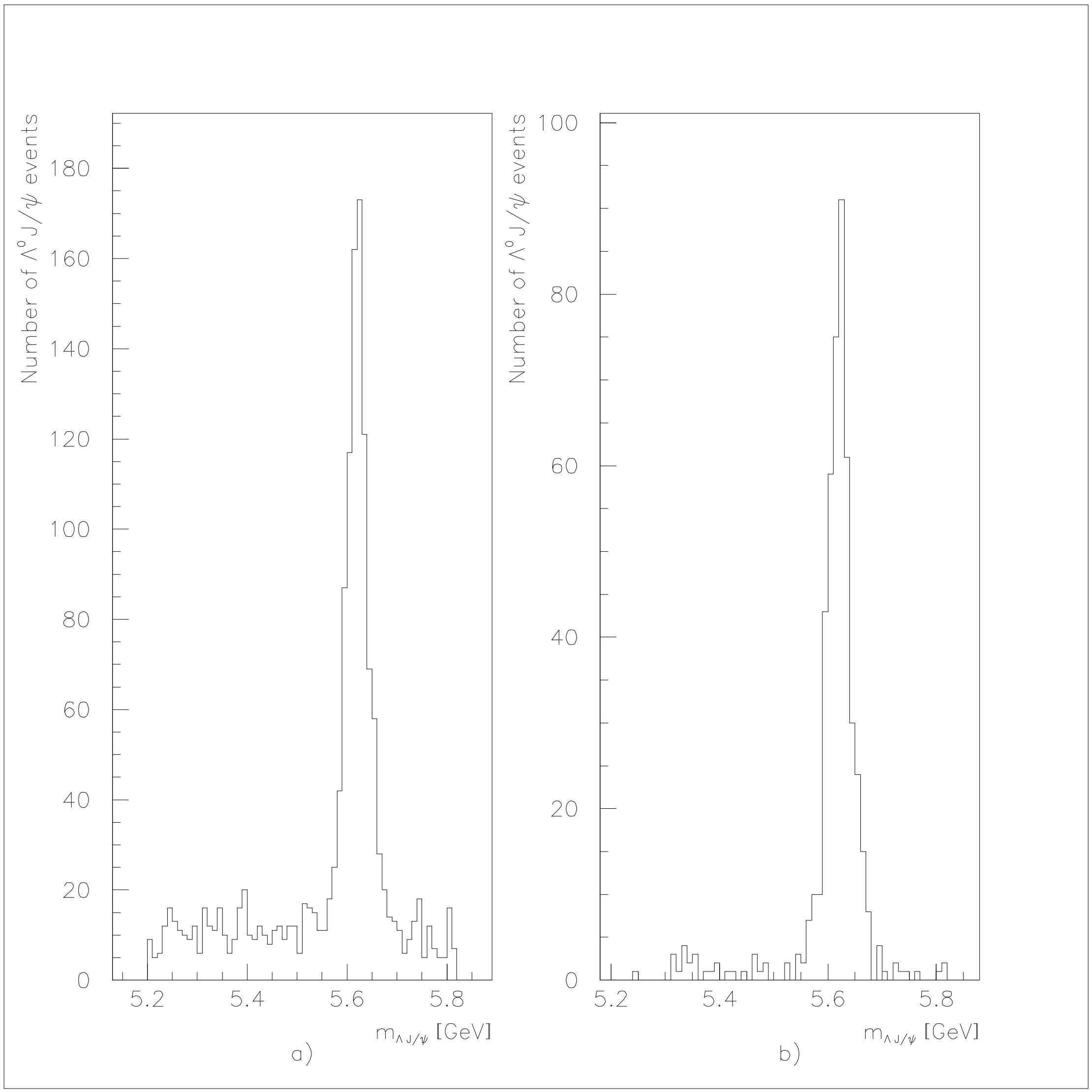

Figure 6.8: The Λ^0 J/ψ effective mass distribution: The peak at 5.62 GeV is from Λ_b^0 and background comes from J/ψ from a b-hadron decay and Λ^0 either from the multiparticle production or from a b-hadron decay (a). The events that passed the cut on the transverse momenta ($p_T > 0.5$ GeV) for p and π^- from Λ^0 decay (b).

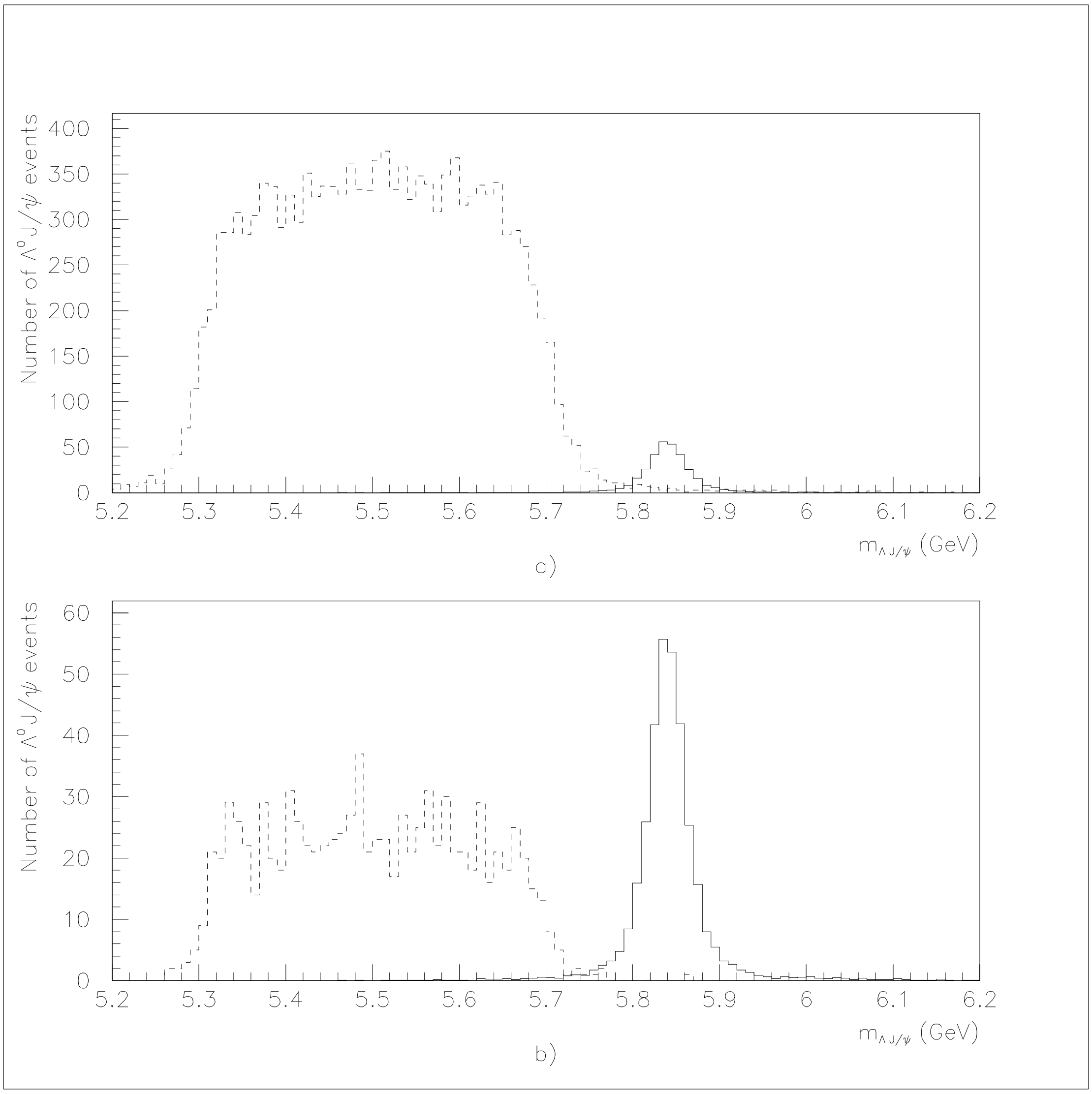

Figure 6.9: The Λ^0 J/ψ effective mass distribution: The peak at 5.84 GeV is from $\Xi_b^0 \rightarrow \Lambda^0\ J/\psi$ decay. The background with the centre at $\approx$ 5.5 GeV comes from $\Xi_b^0 \rightarrow \Xi^0 J/\psi$, $\Xi^0 \rightarrow \Lambda^0\ \pi^0$ and $\Xi_b^- \rightarrow \Xi^- J/\psi$,$\Xi^- \rightarrow \Lambda^0\ \pi^-$ decays (a). The events that passed the cut on the minimal distance of J/ψ and Λ^0 ($d < 1.5$ mm) (b).

Chapter 7

Conclusion

The effort to introduce a heavy weight Software Process into HEP hasn't met with a great success not only in ATLAS. There are general reasons why this kind of Software Process is not completely adequate to HEP, often the same reasons as in software development activities elsewhere. The solution is not to abandon Software process, but to adopt more suitable one. HEP can certainly learn from the experienced gathered elsewhere, namely from the successful adoption of lightweight software processes. Two of them are covered in more details and recommended - Extreme Programming and Feature Driven Development. Also reusable components of the traditional Methodologies are covered, as well as appropriate formulation of the Capability Maturity Model levels. Changing the Software Process doesn't mean denying of the importance of the proper Design Techniques for the practical software development. While ATLAS (and other HEP experiments) have formulated their Coding Rules, the rules (or recommendations) for the software Design are put together for the first time. Here, also a procedure for adopting External Software in the experiment has been formulated.

The complete Architecture for the ATLAS Graphics Framework has been designed and implemented following appropriate software methodologies. The process has started by the Use Cases collection, through the formulation of the major Architectural Principles to the actual Detailed Design and implementation. All those phases are described as well as important parts of the actual existing Framework, namely the Control Structure and the 3D Event Display - GraXML. The Framework is currently in use in ATLAS, reusable components are becoming part of the HEP-wide FreeHEP library and Analysis-Graphics HEPStudio. As the implementation language of the ATLAS Graphics system is Java, note about the role of Java in ATLAS is appended.

The new Pattern Recognition technique based on clustering has been invented.

The study of the Dilution Factor dependency on the ATLAS detector setup on the process $B_d^0 H_b \to J/\psi K^0 \mu X$ have been used for the decisions about the hardware design of the ATLAS detector. The dilution D factor due to tagging is independent on the trigger threshold p_t for $p_t > 6$ GeV. As a consequence, the error on $\sin(2\phi)$ depends on p_t only through rates dN/dp_t.

It has been found that at LHC luminosity 10^4 pb^{-1} the beauty baryons Λ_b^0 and Ξ_b^0 can be measured with the help of angular distributions in the cascade decays $\Lambda^0 J/\psi \to p\pi^- \mu^+ \mu-$ and $\Lambda^0 J/\psi \to p\pi^- e^+ e-$ with the statistical precision better than 0.016 for Λ_b^0 and 0.28 for Ξ_b^0.

Presented results have been published in international physics journals, presented on Conferences and captured in ATLAS Internal Notes as well as ATLAS Technical Design Reports.

Appendix A

On the importance of Java to ATLAS

This Appendix contains the the copy of the ATLAS internal Note about the role of Java in the ATLAS software, which has been co-written by author (with S.Fisher). Author has written Sections A.3 - A.8. Besides being an important contribution to the ATLAS software strategy making, its conclusions are highly relevant to the Software part of the Thesis.

The Java programming language has had a huge impact in the world outside HEP and is now making itself noticed within HEP. Here we discuss why we should consider it, and what the impact might be.

A.1 Introduction

If HEP were to start the migration from FORTRAN to OO today, it is very likely that the choice would be Java.

This does not imply any criticism of the original choice of C++. At the time it seemed sensible; Java had only just appeared as something for building web toys. The world has moved on. The arguments that previously favored C++ now point to Java. Nobody would promote C++ as a safe or easy language, yet Java exhibits both characteristics.

Java appeared about five years ago and is used in a number of recently developed applications. For example most XML tools are Java based. In HEP it has been demonstrated mainly in highly graphical applications such as JAS (Java Analysis Studio) or the WIRED Event Display, but it is also being used for the reconstruction software (hep.lcd) being developed in the LCD collaboration at SLAC.

The main part of this document is made up of lists of Java features which have been categorized as follows:

- **Section 2.** lists those features of Java which make it clearly better than C++
- **Section 3.** also a pro-Java category which explains those features which were handled badly when the language was young, but have now been solved
- **Section 4.** short list of possible problems
- **Section 5.** short list of unresolved problems.
- **Section 6.** lists those features which are neither good nor bad; they are things which Java just does differently.
- **Section 7.** summarizes the potential of Java in HEP.
- **Section 8.** makes specific recommendations to ATLAS concerning the future Java strategy.

A.2 Benefits of Java

One of our collaborators was heard to say "C++ makes writing code much more of a challenge". This is almost certainly true and is the reason why we should move away from C++ - it is needlessly difficult to get right. People should be able to concentrate on the algorithms and not worry about the details of memory management. We want to empower people to be able to contribute towards the ATLAS software effort and so we cannot afford the complexity of C++. It is remarkably easy to become fluent in Java, you may need to look up the details of some class you wish to use but rarely need to check in a textbook for details of the language itself.

A.2.1 Automatic memory management

There is no need to delete objects manually with Java. Like most languages (apart from C++) objects get deleted when they are no longer reachable. This does not completely free the programmer from the burden of thinking about who owns objects as it is still possible to prevent objects being garbage collected by keeping a reference somewhere. However it is not possible to follow a pointer to some piece of memory where an object used to be. If you set a reference to null, and try to use it you get an exception immediately.

A.2.2 No "$*$", "$->$" or "&" operators to worry about just "."

Objects are always passed by reference (and simple data values by value). Object references act as C++ pointers to objects but you use "." instead of "$->$". If you want to copy an object you must explicitly clone it, as assignment just sets a reference to what is on the right hand side of the assignment.

A.2.3 Clean definition of interfaces

Java encourages the programmer to think about interfaces. An interface specifies the set of methods which characterize it. A class may implement many interfaces. Note that by contrast a class may only extend (i.e. inherit from) one class; this avoids the complex C++ rules about what happens when a class appears twice in the inheritance tree.

A.2.4 No "`.h`" files

Keeping headers and code consistent in C++ can be quite painful. In Java there are are no headers and so instead of #including a header file, the compiler is able to obtain the information it needs from the other Java files or from the already compiled files. It is true that .h files make good documentation (except that they also show the private member data) but the information which can be extracted by the Javadoc program, which is a part of the standard Java distribution, is much better.

A.2.5 World-wide naming scheme

Java recommends a package naming scheme which if followed guarantees the world wide uniqueness of every class name. This should mean that you can use any library without fear of name clashes.

A.2.6 No preprocessor

Multi-platform development is very easy because the Java source code will compile on multiple platforms and the resulting byte code will then execute anywhere - this is even true between Linux and NT. C++ has the preprocessor which can handle conditional code (e.g. `#ifdef`) to cope with platform variations. For Java there is no preprocessor and so no conditional code. This makes program testing much simpler.

A.2.7 Byte code can execute anywhere with a JVM

A Java compiler does not produce machine code but portable byte code. The Java Virtual Machine(JVM) is the software which executes the portable Java byte code.

Java is evolving so you should use a recent JVM. You would want to do this anyway to benefit from the performance these JVMs offer. One of the more exciting ways to exploit this feature is with mobile agents. Code compiled on NT really runs properly under Linux and vice versa.

A.2.8 Fast development

The compilers are fast and give clear diagnostics and there is no linking step to be carried out. This results in a very fast program development cycle and faster prototyping.

Code is loaded on demand, so you pay no penalty for executing only a small part of a big program. This is similar to dynamic linking in other programming languages, but with Java it was designed in from the beginning - so it works effortlessly and reliably

A.2.9 Fast execution

With the latest technology from IBM and SUN , as described in Section 3.1, Java programs have performance comparable to their C++ equivalent.

A.2.10 Thread management is part of the language

This may seem to be of limited interest to most of us in the offline. However there are real uses - for example to avoid delays waiting for the next event to be read. For a GUI to behave in a reasonable way threads are essential; otherwise the GUI is blocked while the application part of the program is running.

A.2.11 Exact reproducibility of results

Currently reproducibility of results across hardware is part of the design of the language. For reasons of efficiency it is now possible to turn this off. However the benefit of expecting bit for bit identical results on different hardware should not be underestimated. We would need to check in our applications, if and when it is worth sacrificing reproducibility of results across hardware, in favor of efficiency.

A.2.12 "Reflection" facilities

Reflection is the Java term for the ability of a program to learn all about the objects it contains. This allows you to do tricky things - but still in a type-safe manner. It makes interfaces to databases very elegant and is the underlying support for a lot of the features which are taken for granted with Java - for example excellent (and portable) debuggers, serialization (see Section 2.13 and JavaBeans).

A.2.13 "Serialization"

Any class which implements the Serializable interface can be serialized, i.e. written out to to a file. You just have to put implements Serializable after the class name in the declaration of the class and it can be output with no effort.

A.2.14 Dynamic optimization

A huge amount of effort has been invested in various technologies to make Java execution fast. This work has resulted in claims of better performance for Java than C++. Most of the work centers around efficient garbage collection and upon dynamic optimization. The virtual machine notices where the code is spending its time and optimises that section. A normal JIT compiler converts byte code to machine code the first time it is executed. The newer technologies profile the code as it is executing and decide which parts are worth the cost of converting to optimized machine code.

A.2.15 Dynamic loading

There is no huge executable to build; instead classes are loaded dynamically as they are needed. This also gives extra flexibility in complex systems as the application can decide itself which libraries to use. You can even write your own class loader if you have unusual requirements.

A.2.16 Javadoc generated documentation

By simply following the Javadoc conventions as recommended in ATL-SOFT-2000-002
http://weblib.cern.ch/format/showfull?base=ATLATL&sysnb=0001741 excellent package documentation can be generated automatically.

A.2.17 Excellent tool support

Tool support for Java is excellent with many free and commercial tools available. For example the CASE tool, Together, supports both C++ and Java but new features (such as audits, metrics and patterns) are always released first for Java.

A.2.18 Easy to configure for compilation

To compile Java code you only need to know where the source file (or files) are, and the locations of the already compiled files. The directories with compiled files (or a compressed archive) are normally specified with the CLASSPATH environment variable. The various compilers and IDE tools use this, and small programs have no need for make or similar programs to control the build sequence.

A.3 Non-problems

This section lists the items which are often regarded as problems, but which in reality either are not problems at all or are problems in any programming language because they reflect the difficulty of the problem they are solving.

A.3.1 Performance for numerical operations

A frequent complaint heard about Java is its speed. This often comes from people who just assume "java = interpreted = slow", or from people whose primary experience of Java is waiting for applets to download into Netscape. It has been demonstrated several times, as indicated in the references just below, that intensive numerical calculations in Java are of comparable performance to those in C++. The code which actually runs for a Java program is real compiled code generated by a JIT or HotSpot compiler. The performance dependency on benchmark parameters indicates that the actual executable code is very similar for Java and C++; the same patterns connected to e.g. cache size is observed in both Java and C++ programs. The performance depends much more on the choice of algorithm, libraries and program design than on the language. New HotSpot compilers allow behavior-driven dynamic compilation and profiling of the compiled Java code. While this may make short programs slower due to the profiling overhead, longer programs can be optimized to a level inaccessible to standard techniques used for C++ optimization. Java has already reached the performance of the standard optimized C/C++ programs and it approaches the performance of programs designed for performance in C/C++/F77.

- **IBM's Java performance research**
  http://www.research.ibm.com/journal/sj39-1.html

- **C vs. Java speed comparisons**
  http://www.aceshardware.com/Spades/read.php?article_id=153

- **Java versus C++**
  http://www.multimania.com/lefevre/java/ceaxx.html

- **Java Performance**
  http://hepunx.rl.ac.uk/hep2java/meeting/16-03-2000/hoschek2/sld001.htm

- **Java Performance**
  http://atlas.web.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Taylor-00.05.12/

- **Java Grande Forum**
  http://www.javagrande.org/

- **Java for Scientific Computing**
  http://www.npac.syr.edu/projects/javaforcse/

- **The Java HotSpot Performance Engine Architecture**
  http://java.sun.com/products/hotspot/whitepaper.html

A.3.2 3D graphics performance

The speed and quality of 3D graphics in Java applications used to be inferior to that of applications based on C/C++ with OpenGL/OpenInventor as the foundation libraries. Java3D and OpenInventor both use OpenGL/Mesa as the mechanism for the actual 3D drawing so can both profit to the same extent from the hardware acceleration available from modern PC video-cards. Coupled with the performance improvements of the JVM, applications built on top of Java3D now have similar performance to those built in C/C++ with OpenInventor. The final performance depends primarily on the design and implementation of the visualization libraries and programs.

- **Java3D**
  http://java.sun.com/products/java-media/3D/index.html

A.3.3 Numerics

Java is designed to give exactly reproducible results (including floating point operations) on all platforms, all compilers and all Virtual Machines. This feature is very important when results from variety of platforms should be combined. However, there are certain tradeoffs connected to it as hardware features available only on some platforms can't be always fully exploited. This may sometimes lead to the loss of speed and precision.

Intel processors (among others) allow Double-Extended (80bit) precision for floating point numbers. To guarantee exact reproducibility, Java rounds such number into standard Double (64bit) precision. This leads to the loss of precision and speed in performing the rounding operation. For the same reason, Java doesn't allow fused multiply-accumulate which slows some operations such as matrix manipulations. The exact reproducibility has also certain optimization consequences: for example the compiler is not free to reorder associative operations.

The problem of not being able to fully exploit the hardware has been mostly solved by Java 1.3, which can internally used float extended-exponent and double extended-exponent. A new keyword, `strictfp`, has been introduced to enforce exact reproducibility (which is now not the default). Java also contains two mathematical libraries, `java.lang.Math` (for precise and fast mathematics) and `java.lang.StrictMath` (for exactly reproducible mathematics). Any remaining performance problems connected with the exact reproducibility have been largely outweighed by the superb numerics built in to the new JDK 1.3 from IBM.

Java doesn't allow access to the rounding mode and only does "round to nearest" (IEEE 754 default). However in C++, rounding mode is implementation specific. Access to all IEEE 754 modes may be available, but portable code can't rely on it.

Java doesn't have flags for IEEE 754 Exceptions: Invalid Operation, Overflow, Division-by-Zero, Underflow and Inexact Result. So the program can't handle them easily. Instead the program takes some default action which may not be always optimal. Java only defines one NaN (not-a-number), while IEEE 754 specifies several. C++ may turn floating point exceptions into C++ exceptions, but again it is implementation specific.

In summary the numeric problems have either been solved in the current release of the Java specification, or the C++ situation is no better than the Java one. C++ implementations may give wider control over the processing hardware, but Java always allows the user to choose between exactly reproducible results or to use the hardware to optimize speed and precision.

- **How Java's Floating-Point Hurts Everyone Everywhere** http://www.cs.berkeley.edu/ wkahan/JAVAhurt.pdf

- **Updates to the Java Language Specification for JDK Release 1.2 Floating Point**
  http://java.sun.com/docs/books/jls/strictfp-changes.pdf

A.3.4 Memory Management

Automatic garbage collection is another strong point of Java. The limited possibility to control it is often mentioned as a problem for larger Java programs. Memory management can be optimized in C++ but there is no control in Java. Java doesn't allow the developer to cluster objects efficiently in memory.

Sun's HotSpot compiler contains advanced memory allocation and garbage collection mechanisms, which is in most cases more effective than anything done by hand by a knowledgeable C++ programmer or by an add-on C++ garbage collector. If a user really needs to control when the collection happens, he or she can trigger it so as to reduce the likelihood of it having a major impact at the wrong moment in an interactive program. This possibility is, however, very rarely needed as the standard HotSpot garbage collector collects and re-clusters memory fluently. Weak references offer another way of efficiently interacting with the object management system and allows implementation of sophisticated ways of object caching.

- **The Java HotSpot Performance Engine Architecture**
  http://java.sun.com/products/hotspot/whitepaper.html

A.4 Possible problems

These are problems which may be concealed in not-yet fully tested features or in application domains where Java has not yet been tried.

A.4.1 Handling of too many objects

There is a need to test how Java behaves with many small objects. Tests indicate that the HotSpot memory manager performs especially well in those cases and is able to prevent memory fragmentation.

A.4.2 Running in the Event Filter and Level 2 Trigger

Java should be tested in the Event Filter and Level 2 Trigger to ensure that it meets the performance requirements. No significant problems connected to the performance or memory management are expected.

A.5 Unresolved problems

These are real problems for which a solution should be found.

A.5.1 Access to databases

Java can either access objects in a database based on C++ (where C++ objects should be converted to the Java objects on-the-fly) or directly in a Java database. While there are candidates for a suitable database using C++ technology, experience with Java-based databases is much smaller. Some promising candidates (see below) for a fully-featured Java-based OO database suitable for HEP have appeared which seem to be quite close to satisfying our functional requirements but performance and scalability are not known.

The experience from other areas, such as CASE tools, shows many small-scale applications appearing which reflects a real need for that application. This is shortly followed by mature usable applications. This is exactly what is happening today with Java databases: the range of candidate OO databases is very wide from the simple Open Software products such as Ozone, through the Java binding of the traditional OO databases such as Objectivity to the new Java-specific OO database solutions such as JDO and Forest.

- **Objectivity Java Binding**
  http://www.objy.com/Products/Java/Objy5JAVA.htm

- **Ozone**
  http://www.ozone-db.org/

- **JDO**
  http://java.sun.com/aboutJava/communityprocess/review/jsr012/JDO_0_8.pdf

- **Forest**
  http://www.sun.com/research/forest/

A.5.2 Java $< - >$ C++ collaboration

It is generally difficult to interface different OO languages. As some HEP code has already been written in C++ and more code may be written, the question of interfacing between C++ and Java is frequently seen as the major obstacle for introducing Java into the HEP environment on a wide scale. While this argument may be used to advocate the most rapid possible migration to Java, it is still worth considering the Java ¡-¿ C++ inter-operation to make use of significant C++ packages for example.

Various possible language inter-operation scenarios have been investigated as indicated in the reference below. The most advanced seems to be the JACO project, aiming at the transparent access of Java to C++ (and vice versa). Experience accumulated so far in HEP indicates that the language inter-operation could be automated provided that strict interfaces are defined between those packages which must communicate and are written in different languages. Having a well defined simple interface is good practice, so having packages in different languages may be of some benefit as it forces us to define interfaces clearly.

- **Meeting on HEP migration to Java - C++/Java interworking**
  http://hepunx.rl.ac.uk/hep2java/meeting/25-05-2000/

A.5.3 Non-existence of immutable objects

There is no equivalent of the C++ const keyword in Java. There is only `final`, which is very different from const and can rarely be used as an equivalent. In particular there is no keyword to mark formal parameters as immutable.

The problem could be handled by the use of naming and coding conventions. It may turn out that this problem will disappear once people become accustomed to the fact that they are dealing with the object rather than a copy of it and will then treat it with care. Further protection can be offered by ensuring that any "set" methods only have package visibility so a user from another package cannot do any harm.

Another technique is to define "const interfaces" for objects which are not to be modified. The idea of a const interface, which a class can implement, is that it only exposes methods which return simple data and const interfaces to other objects. A const interface is not generic but must be defined for each class; for example ConstEvent, ConstTrack and ConstHit. Note that these are not extra objects but just safe interfaces to existing objects.

A.6 Other features

Those are just the features of the language. They may sometimes be seen as benefits or problems.

No multiple (non-abstract) inheritance.

This is very often the sign of a good OO style even in other OO languages.

No operator overloading.

For non-primitive types, operator can't be overloaded. For example, vectors can't be added as A + B, but only as A.plus(B). There are attractions in allowing operator overloading to support mathematical operations where the notation is close to the normal mathematical formalism and one can take advantage of the operator precedence rules.

No automatic type narrowing.

The type conversion which could lead to a loss of precision doesn't happen automatically, but variables must be cast explicitly. This protects the developer from certain hard-to-find bugs and makes the behavior of the code explicit, but some may find it annoying.

No genericity.

Unrestricted genericity, where the type is an argument to a class to be used when the object is instantiated, as implemented for example in C++ as templates, represents a programming paradigm largely orthogonal to OO. Very often, C++ templates are not used for generic programming but to solve other C++ problems (for example non-existence of reflection, so the programmer should supply the type information which could otherwise be extracted from the object itself). Compilation of C++ templates still poses serious problem for some compilers. In Java, most of the C++ templated constructs can be rewritten using reflection, Class and Object objects.

The place where templates are useful is in constructing typed collections.

The standard Java collections will hold anything. It is only when you get the object and cast it to the type you expect that you discover it is not what you had anticipated. However it is easy to write a wrapper class which will look almost exactly like an STL container; you can constrain the type of object it will hold by passing an object or class to the constructor of the collection.

No macros.

The only legitimate use of macros in C++ is to solve the platform dependency problem (Section 2.6). This means that macros are not needed in Java.

No global functions.

They are not needed in the real OO language. Also static methods provide the same functionality without taking global name space.

No inlined functions.

Even in C++, the compiler often does a better job than the programmer.

No typedef.

The typedef in C++ is normally used to hide the complexities of generic collections. This is clearly not a problem for Java.

No enum.

It would be nice to have enum. To get a similar effect you can define a series of static final ints but you have to look after the values yourself.

Arrays are only one dimensional.

This is not too serious with the new highly optimized mathematical libraries from IBM. You can have arrays of arrays. This can be used to represent triangular matrices very directly.

Array bounds and null pointer checking.

While these features cause certain performance degradation, it makes program execution much safer. Exceptions (from out-of-range indexes and null pointers) happen immediately and are easy to pinpoint. In C and C++, the checking should either be implemented by a user or program crashes later at a random point. New Java compilers can to a large extent remove the performance problem by analyzing the code and identifying which checks are superfluous.

String is 16bit Unicode (not 8bit ASCII).

Strings are not used so often in HEP to create problems with space. The advantage of the Unicode is easy internationalization and easy access to non-Latin-1 characters (like Greek alphabet).

Java isn't (ISO) standard.

The current Java Community Process works fine in introducing new useful features while granting reasonable language stability.

A.7 Place for Java in HEP

Java is emerging as the best general purpose computing language available at the moment and as such we expect it to become the 'default' language for most tasks in HEP.

A.7.1 Framework

Java's features (natural dynamic loading, platform independence, network awareness, reflection,...) make it ideal for the building of the Framework. The Framework should allow other languages as algorithms or services. We should be ready for other languages which will become important but which may not have been invented yet. Java is already being successfully used for the Framework (and all its components) in the work for the LCD experiment.

A.7.2 Reconstruction

As there's no performance problem in Java and the algorithmic code is almost identical in Java and C++, reconstruction algorithms can easily fit into the Java environment. Algorithms written in either language should be able to transparently collaborate. The "ordinary physicist" is especially involved in the reconstruction and simulation and even more so in the analysis. The benefits of coding in a robust language are immense in these areas.

A.7.3 Simulation

The current mainstream simulation strategy in HEP is based on the Geant4, which is written in C++ and it's unrealistic to expect it to be re-written in any other language soon. Also the key particle generators are written in various other languages (Fortran, C++) and some of them will stay as they are. For those reasons, it's clear, that if Java is massively used in HEP, Simulation should be reasonably interfaced. This should, however, not be very difficult as the interface between simulation and other domains is clearly defined.

A.7.4 Database

Java offers a natural basis for the building of OO databases because of features such as serialization and reflection. There are already several promising OO databases products or prototypes, which may eventually satisfy HEP needs. This area needs further study.

A.7.5 Analysis and Graphics

Most of the mature Analysis packages such as JAS, WIRED, Atlantis and GraXML, are already written in Java or are in the process of migration. These programs are able to cooperate or stand alone. It seems very natural to use Java as the primary language for analysis.

A.7.6 User Interface

Java is recognized as having been a success for the GUI. Swing is very easy to use directly and there are many tools (in Java of course) which allow you to build your GUI graphically. There used to be a problem with speed of the Java graphics, this problem has been, however, already successfully solved except that Swing is still a bit slow. There is also the possibility to use Java via a command-line interface which makes it a good candidate as a scripting language (or as the interface to other scripting languages).

A.7.7 Grid

Java has been developed from the beginning as the language for building network-wide distributed client-server applications. Many of its features are directly aimed at the development of the application running transparently over the network (security, platform independence, remote message invocation, applets, servlets, mobile agents). If any language fits into the Grid environment, then it's Java. It is also worth noting that the Globus toolkit which many people are considering as the basis for Grid middleware is in ANSI C with a few Java tools around it. They avoided C++ for reasons of portability.

A.8 Recommendations

In general, we can say that there are no serious obstacles to introduce Java as the main implementation language for the HEP software development. All frequently claimed Java problems are either not problems at all, or they are already solved. At the same time many of the Java advantages will have clear positive impact on the functionality, performance and maintainability of our software. Also the number of existing standard and mature Java applications and class libraries will make the future software development for HEP much faster, safer and easier.

In a mixed Java-C++ environment, one can't fully exploit all the Java advantages. So it would be preferable to replace C++ by Java to benefit from OO without fighting the implementation language. Common experience shows, that once programmers start using Java, they won't go back. It appears that the time is right to start the move towards Java and we should not delay further.

A.8.1 Architecture

The ATLAS framework should be extended to allow transparent inter-operation of Java and C++ Algorithms with transparent access to Services in either language. Efficient access to (transient) data storage should be available to both languages.

The ATLAS coding conventions should be amended to discourage C++ features which may create problems for Java$< -- >$C++ inter-operation and migration to Java. The most important is using non-abstract multiple inheritance and generic programing using templates.

A.8.2 ATLAS Support

Java should be fully supported in ATLAS. In particular, all standard tools including release tools and CASE tools, should be able to handle packages fully written in Java and some packages written in Java and others in C++.

A.8.3 CERN Support

ATLAS should request full support for Java from CERN IT. This includes installation and maintenance of JVMs and other needed Java libraries and applications as well as active development and support of key HEP Java packages, such as WIRED, JAS and Colt.

ATLAS should request a detailed study of the Java solutions and interoperability for software products developed or supported by CERN/IT. This concerns mostly LHC++, RD45 and GEANT4.

A.9 Acknowledgements

In this note, text from Documentation, Web Pages and Papers of the following authors has been used: Mark Donszelmann, Wolfgang Hoschek, Tony Johnson

Valuable information, on the finer points of C++, has been provided by Wim Lavrijsen.

A.10 Some useful Java products

This is just a short list of some Java products (potentially) useful for the software development in HEP. Description are mostly taken directly from the package description. The list is by no means exhaustive.

A.10.1 HEP

There are already several mature HEP products covering mostly the Domains of data analysis and basic computing infrastructure. Very often, those products are of higher standard than corresponding packages in Fortran/C/C++.

The Colt Distribution

This provides an infrastructure for scalable scientific and technical computing in Java. It consists of several free Java libraries which have been repackaged for convenience along for user convenience bundled under one single uniform umbrella. Some of its man features are listed below:

Templated Lists and Maps. Dynamically resizing lists holding objects or primitive data types. Operations on primitive arrays, algorithms on Colt lists and JAL algorithms can freely be mixed at zero copy overhead. Automatically growing and shrinking maps holding objects or primitive data types. Space efficient high performance BitVectors and BitMatrices. Templated Multi-dimensional matrices Dense and sparse fixed sized 1,2, 3 and n-dimensional matrices holding objects or primitive data types. Linear Algebra Standard matrix operations and decompositions. LU, QR, Cholesky, Eigenvalue, Singular value. Histogramming. Compact, extensible, modular and high performance histogramming functionality. Mathematics Tools for basic and advanced mathematics: Arithmetics and Algebra, Polynomials and Chebyshev series, Bessel and Airy functions, Constants and Units, Trigonometric functions, etc. Statistics Tools for basic and advanced statistics: Estimators, Gamma functions, Beta functions, Probabilities, Special integrals, etc. Random Numbers and Random Sampling. Strong yet quick. Partly a port of CLHEP. JAL A partial Standard Template Library port by Silicon Graphics. util.concurrent Efficient utility classes commonly encountered in parallel & concurrent programming.

See: http://nicewww.cern.ch/ hoschek/colt/index.htm

JAS

Java Analysis Studio (JAS) is a graphical application for analysis of high-energy physics data. The application is independent of any particular data format, so that it can be used to analyze data from any experiment. The application features a rich graphical user interface (GUI) aimed at making the program easy to learn and use, but which at the same time allows the user to perform arbitrarily complex data analysis tasks by writing analysis modules in Java using the built in editor/compiler. The application can be used either as a stand-alone application, or as a client for a remote Java Data Server. The client-server mechanism is targeted particularly at allowing remote users to access large data samples stored on a central data center in a natural and efficient way. JAS is written entirely in Java and will run on any platform with a Java virtual machine..

Home:
http://www-sldnt.slac.stanford.edu/jas/

In ATLAS:
http://atlas.web.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/JAS/

WIRED

WIRED is a framework, written in Java, to build High Energy Physics event displays that can be used across the network. Originally WIRED was developed to better understand the applicability of the Java technology in HEP. Now, it has grown to be a framework in use and under development in several experiments, both inside and outside CERN (ATLAS, CHORUS, DELPHI, LHCb, BaBar, D0 and ZEUS). To guarantee portability across all platforms WIRED uses the Java language and its Swing user interface component set. The graphical user interface allows for multiple views and for multiple controls acting on those views. A detector tree control is available to toggle the visibility of parts of the events and detector geometry. XML (Extensible

Markup Language), RMI (Remote Method Invocation) and CORBA loaders can be used to load event data as well as geometry data, and to connect to FORTRAN, C, C++ and Java reconstruction programs. A special Java interpreter allows physicists to write small scripts to interact with their data and its display.

Home:
http://wired.cern.ch/

In ATLAS:
http://atlas.web.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/WIRED/

JACO

Java Access for C++ Objects. Package being developed mostly by JAS and WIRED developers from requirements taken (among others) from ATLAS.

FreeHEP

The goal of the FreeHEP library is to encourage the sharing and reuse of Java code in High Energy Physics. Although some of the code is fairly specific to HEP, other code is more generic and could be used by anyone. To maximize reuse the dependencies between various packages in the FreeHEP library are kept to a minimum, so you can use which ever parts interest you. The FreeHEP Java library is an "Open Source" library distributed under the terms of the LGPL.

See: http://java.freehep.org/lib/freehep/doc/web/index.html

A.10.2 Non HEP

The amount of existing high-quality java code is enormous.

Xerces/XML4J

XML Parser for Java is a validating XML parser written in 100% pure Java. The package contains classes and methods for parsing, generating, manipulating, and validating XML documents.

See: http://xml.apache.org/xerces-j/index.html

LOG4J

With log4j it is possible to control logging at runtime without modifying the compiled code. The log4j package is designed so that logging calls, which have been disabled by the control file, do not incur a heavy performance cost. This is a small, but well designed package.

See: http://www.log4j.org/

BeanShell

BeanShell is a small, free, embedable, Java source interpreter with object scripting language features, written in Java. BeanShell executes standard Java statements and expressions, in addition to obvious scripting commands and syntax. BeanShell supports scripted objects as simple method closures like those in Perl and JavaScript™.

You can use BeanShell interactively for Java experimentation and debugging or as a simple scripting engine for you applications. You can call BeanShell from your Java applications to execute Java code dynamically at run-time or to provide scripting extensibility for your applications. Alternatively, you can call your Java applications and objects from BeanShell; working with Java objects and APIs dynamically. Since BeanShell is written in Java and runs in the same space as your application, you can freely pass references to "real live" objects into scripts and return them as results.

See: http://www.beanshell.org/intro.html

DynamicJava

DynamicJava DynamicJava is a Java source interpreter with a functionality similar to the BeanShell. It is free and distributed with the source code.

See: http://www.inria.fr/koala/djava/

Ozone

Ozone is a fully featured, object-oriented database management system completely implemented in Java and distributed under an open source license. The main goal of the ozone project is to evolve a technology that allows developers to build pure object-oriented, pure Java database applications. Just program your Java objects and let them run in a transaction based, database environment.

See: http://www.ozone-db.org/

Java3D

The Java 3D API is a set of classes for writing three-dimensional graphics applications. It gives developers high level constructs for creating and manipulating 3D geometry and for constructing the structures used in rendering that geometry.

See: http://java.sun.com/products/java-media/3D/

X3D

X3D (Extensible 3D) is an inter-operable set of lightweight, componentized 3D standards. It incorporates a number of component specifications allowing extremely lightweight applications to be deployed on a variety of platforms. Initial components include a lightweight 3D runtime engine with state-of-the-art rendering capabilities, a platform-independent 3D file format and advanced XML integration.

See: http://www.web3d.org/x3d.html

JDOM

JDOM offers an easy to use Java API to manipulate XML documents. While JDOM inter-operates well with existing standards such as the Simple API for XML (SAX) and the Document Object Model (DOM), it is not an abstraction layer or enhancement to those APIs. Rather, it seeks to provide a robust, light-weight means of reading and writing XML data.

See: http://www.jdom.org/

JBuilder

JBuilder is one of many IDEs for developing Java programs. It is quick to learn, and is free in its basic form and offers very good debugging facilities and a GUI builder.

See: http://www.inprise.com/jbuilder/

Appendix B

Graphics User Requirements

This Appendix contains the list of the accumulated User Requirements for the Graphics Domain, which are described in the Section 5.4. All those requirements were accumulated by the author from his own experience and discussions with domain experts.

GUI *(General Functional Requirement)*

Views should satisfy all requirements for GUI.

Graphics should perform as a GUI. It should enable everything, which is required from GUI (but not all Views will fulfill every GUI requirement) and it should provide interface to GUI (in this sense particular GUI implementation can be seen as just a kind of View).

Errors *(General Functional Requirement)*

It should be possible to visualize errors.

Errors means uncertainties here. They may be represented for example by color or thickness of line, by error-bar or error-area,...

Library of objects *(General Functional Requirement)*

Library of high-level objects should be available.

User should have higher level graphical objects like "calorimeter cell" available for constructing his graphical objects. He should not be required to construct them from the basic graphical primitives.

Properties *(General Functional Requirement)*

At least following properties should be available:

1. Colors (ambient, diffuse, specular, emissive)

2. Shininess
3. Transparency
4. DrawStyle (fill, lines, points, invisible)
5. PointSize
6. LineWidth
7. LinePattern

Interactive change of Properties *(General Functional Requirement)*

It should be possible to change visual properties of the graphical objects interactively.

Cameras *(General Functional Requirement)*

At least following cameras should be available:

1. Orthographic
2. Perspective

Genealogy *(Functional Requirement)*

Genealogy of Objects should be visible.

It should be possible to navigate through Object dependencies, so one sees, for example, all Hits for a Track.

Overlay *(General Functional Requirement)*

It should be possible to overlay graphical objects on a View.

Useful for complicated movable patterns.

Framing *(General Functional Requirement)*

Framing of the graphical objects should be possible.

Framing is the drawing a frame around small objects. It enhances effective resolution.

Text *(General Functional Requirement)*

It should be possible to add a text to the View.

For commenting.

Static, streaming and removable objects *(General Functional Requirement)*

It should be possible to define objects which are static, streaming or removable.

Static objects are objects, which are the same for each event, so they can be kept when going from one event to another (e.g.: detector geometry). Streaming objects are objects, which are different for each event, but should be kept as they are used together (e.g.: histograms). Removable objects are objects, which are specific for each event, so they can be discarded when events are changed (e.g.: event display).

Shapes *(General Functional Requirement)*

At least following shapes should be available:

1. Cube
2. Cone
3. Sphere
4. Cylinder
5. PointSet
6. LineSet
7. FaceSet
8. TriangleSet
9. QuadMesh
10. Text2D
11. Text3D
12. Billboard

Rendering techniques *(General Functional Requirement)*

At least these rendering techniques should be available:

1. Wireframe
2. Solid
3. Hidden line removal
4. Flat shading
5. Smooth shading

6. Texture mapping

The rendering techniques can be mixed within one view. For example, Wireframe may be drawn on top of solid.

Objects viewability *(Existential Functional Requirement)*

At least following objects should be viewable:

1. Magnetic field
2. Detector components
   (a) Active detector parts
   (b) Passive detector parts
   (c) Detector status
3. Event data
   (a) Hits
   (b) Trajectories
   (c) Jets

Any Real Objects should have the possibility to be viewable, but some objects are required to use this possibility. The list is not exhaustive.

Ntuple availability *(Existential Functional Requirement)*

Ntuples should be available.

Trees availability *(Existential Functional Requirement)*

Trees should be available.

Context sensitive actions *(General Functional Requirement)*

Context sensitive actions on Real Objects should be available.

There should be a way of using directly properties of Real Objects (member data and functions in case of C++) from the Views.

Creation *(General Functional Requirement)*

It should be possible to create new Real Objects from Views.

Operations on selections *(General Functional Requirement)*

It should be possible to perform operations on selections of real objects selected from the View by picking or by selection language.

Event surfing *(General Functional Requirement)*

It should be possible to navigate through the events forward, backward and by associations with other events.

Event selection *(General Functional Requirement)*

It should be possible to select events by some search or selection mechanism.

Selected events may be directly analysed or the result of the selection may be stored (either in the current data repository or in the new one). It should be possible to store the selection mechanisms.

Restrictable access *(General Functional Requirement)*

It should be possible to restrict access to objects.

Interactivity *(General Functional Requirement)*

All operations should be available in the interactive environment.

Batch *(General Functional Requirement)*

All operations should be available in the batch environment.

Data sources choice *(General Functional Requirement)*

The data source should be choosable by the user.

Shared data access *(General Functional Requirement)*

The data should be accessible by several users simultaneously. It should be possible to lock currently analysed data.

The lock should be available to allow changing of shared data.

Event related information *(General Functional Requirement)*

Event related information should be available.

Event number, date, trigger tags, processing history,...

Parallel Processing *(General Functional Requirement)*

Parallel processing should be possible.

Also over the network by using for example mobile agents.

Presentation graphics *(Existential Functional Requirement)*

Interface to presentation graphics tools should be available.

Picking *(Existential Functional Requirement)*

It should be possible to pick an object in a View.

The picked object should be highlighted on other Views.

Inverse Picking *(Existential Functional Requirement)*

Inverse picking should be available.

Inverse picking is moving the pointer to any pre-selected object by giving its identification.

Histogram operations *(Existential Functional Requirement)*

It should be possible to perform at least these operations on histograms:

1. addition and subtraction
2. multiplication and division
3. application of user-supplied function
4. fitting to user-supplied function

These operations should handle errors correctly.

Ntuple operations *(Existential Functional Requirement)*

It should be possible to perform at least these operations on ntuples:

1. the same as for histograms
2. bin-width changing
3. bin-offset changing

These operations should handle errors correctly.

Histograms from Ntuples *(Existential Functional Requirement)*

It should be possible to generate histograms from ntuples.

Using user-supplied functions and queries.

Command line and GUI *(General Functional Requirement)*

Command line and GUI should be coherent.

So that, for example, Menu updates when command creates different state.

Command abbrevitations *(General Functional Requirement)*

Commands should be available in both full and abbreviated form.

Command line Scrolling *(General Functional Requirement)*

It should be possible to scroll through already executed commands and re-issue them.

Logfile *(General Functional Requirement)*

All actions should be recorded in the logfile in a form which performs replay.

Output Compactness *(General Functional Requirement)*

Output to terminal should be compact.

Output and Input *(General Functional Requirement)*

Program output and operator input should be separated.

Command line *(Existential Functional Requirement)*

Command line user interface should be available. It should provide access to all software functionality.

GUI *(Existential Functional Requirement)*

The Graphical User Interface should be available. It should provide access to all major software functionality. It should not be required to run the software.

Specialized GUIs may exist, with only subset of functionality (with restrictions).

Echo *(Existential Functional Requirement)*

All command should be echoed (on demand).

Also when issued from GUI.

Wizards *(Existential Functional Requirement)*

Software Wizards should be available.

To make learning easier and to make complicated tasks shorter.

GUI visibility *(Existential Functional Requirement)*

GUI should be visible all the time, but it should not disable using of other graphical Views.

The GUI window should not compete with the pictures windows, but they should be visible simultaneously.

Help *(Existential Functional Requirement)*

Topics related help should be available.

Scripting *(Existential Functional Requirement)*

Scripting language should be available.

Numerical Values *(Existential Functional Requirement)*

Numerical values should have current settings starting with valid defaults.

So that only those values should be given, which need modification.

Objects grouping *(General Functional Requirement)*

It should be possible to perform operations on created groups of objects.

Operations will be performed on Graphical Objects directly and on Real Objects via their connections to Graphical Objects.

Objects hiding according to type *(General Functional Requirement)*

It should be possible to hide/un-hide objects according to the type of the object.

Subviews *(General Functional Requirement)*

It should be possible to show subset of Objects.

Possibly defined by their real features.

Objects hiding according to visual size *(General Functional Requirement)*

It should be possible to hide/un-hide objects according to the visual size of the object.

Viewpoints changing *(General Functional Requirement)*

It should be possible to change viewpoints.

Changing viewpoint means jumping over a set (previously defined or new) points in space. One will change distance and may leave something behind.

Visual Operations availability *(General Functional Requirement)*

At least following Visual Operations should be available:

1. Rotation around any axis
2. Zooming by general 2D transformation

3. Zooming in any direction
4. Zooming with constant direction, angle or aspect ratio
5. Translation
6. Scaling
7. Scaling in any direction
8. Lorentz transformation
9. Magnification of the selected area

The zoom area is selected by rubber-band.

Zero approximation *(General Functional Requirement)*

Zero approximation view should be available.

Visual properties *(General Functional Requirement)*

It should be possible to define visual properties of objects.

Visual properties are e.g. color, line type, fill pattern, e.t.c.

Client-Server *(General Functional Requirement)*

The system should be able to run as a client-server application using network.

Data on views coherence *(General Functional Requirement)*

Same views of the different data should be coherent. It should be possible to switch this feature off.

Textual information *(General Functional Requirement)*

Textual information associated with graphical objects should be available.

Views coherence *(General Functional Requirement)*

Different views of the same data should be coherent. It should be possible to switch this feature off.

Change in one view should automatically propagate into other views.

Corresponding Pointers *(General Functional Requirement)*

It should be possible to show corresponding pointers on different Views.

Probably using cross-hair pointers.

Views multiplicity *(General Functional Requirement)*

It should be possible to have several views simultaneously.

Possibly with different events or selections.

Histograms and Event Display connection *(General Functional Requirement)*

It should be possible to pass information and control between Event Display and Histograms.

Textual Views availability *(General Functional Requirement)*

Textual output should be available.

Event Display *(Existential Functional Requirement)*

Event Display should be available.

Correlations *(Existential Functional Requirement)*

It should be possible to show correlations between objects.

For example by using colors.

2D Event Display Views availability *(Existential Functional Requirement)*

At least following Event Display 2D views should be available:

1. x-y
2. rho-z
3. phi-rho
4. phi-z
5. phi-theta
6. phi*sin(theta)-theta
7. pt-pz
8. eta-pt
9. eta-phi
10. y'-z
11. Fish-eye x-y
12. Fish-eye rho-z
13. Clock transformation
14. V-plot

15. Skew-plot
16. Hierarchy
17. 2D slice of 3D
18. User defined f-g
19. u-v
20. f(u,v)-g(u,v)
21. Text dump

*The 2 basic pictures are x-y and rho-z due to a) the cylindrical shape of the detector, b) the solenoidal field parallel to the z axis and the toroidal field circular around the z axis, c) the structure of the event which is independent of phi, d) these two (orthogonal) projections show all properties of tracks originating from the center. Due to the cylindrical structure of the detector, magnetic field and event, the angular projections phi-rho, phi-z and phi-theta should be available. Due to the central role of the z axis, y'-z projection should be available with y'=x*cos(s)+y*sin(s), where s is defined interactively. u,v are coordinates measured in 2D detector. Corresponding to the high space resolution of small inner subdetectors and low space resolution of the large outer subdetectors, circular and rectangular Fish eye transformation for x-y and rho-z should be available. V-plot is used to distinguish tracks. Text dump should contain objects with attributes.*

Kinematic Space *(Existential Functional Requirement Views)*

It should be possible to make Event Display views for both spacial and kinematic spaces.

Kinematic space contains directions, 3 vectors, 4 vectors.

Geometrical cuts *(Existential Functional Requirement)*

It should be possible to make geometrical cuts.

With or without the consequences on other views.

Histogramming *(Existential Functional Requirement)*

The histogramming View should be available.

Graphical output formats *(General Functional Requirement)*

Output in widely used file formats should be available.

E.g. wrl, gif, ps, xml.

Progressivity *(General Functional Requirement)*

Slow operations should be progressive.

The most important parts of the view should be displayed first. Histograms should be filled progressively. There should be a visual Progressivity indicator. The granularity may change during the longer phase of displaying. "Slow" means slower than is usual in overall computing environment.

Interuptability *(General Functional Requirement)*

Slow operations should be interuptable.

There should be a visual indicator of the running operation.

Printing *(General Functional Requirement)*

It should be possible to print (selected regions of) views.

Ability of creating PostScript files will satisfy this requirement.

Setup change *(General Functional Requirement)*

It should be possible to change setup.

Setup means here all properties (options, preferences) of the View.

Setup saving *(General Functional Requirement)*

It should be possible to read/save setup.

Persistency *(General Functional Requirement)*

It should be possible to make View persistent.

User should be able to restart a View in the same state as before exit. Modified or created data should be kept. In case of modifications, original version should be also available.

Applications and Libraries *(General Functional Requirement)*

The system should be able to run as stand-alone application but also provide means for integration with the user software environment.

Speed *(General Functional Requirement)*

Views should be reasonably fast.

Rough estimates for the speed limits are:

- *Analysis and production of hardcopies* - 1*s*.

- *Processing of a hardcopy file* - $10 - 20s$.
- *Drawing for smooth movements* - $40ms$.

Extensibility *(General Functional Requirement)*

It should be possible to extend the environment with the user-supplied code.

Standard setup *(General Functional Requirement)*

Standard setup should be available.

Storing views *(General Functional Requirement)*

It should be possible to read/save selected views (with objects and created new objects).

Interactivity *(General Constrain)*

Event Display shall at all times be capable of reflecting the current state of the Real Objects.

New objects *(General Constrain)*

It should be possible to introduce new objects without redesigning the package and if possible without changing and recompiling the existing code.

User shall be able to add viewability to any of his Real Objects. This will introduce also new graphical objects.

New views *(General Constrain)*

It should be possible to define new views without redesigning the package and if possible without changing and recompiling the existing code.

No special-purpose hardware *(General Constrain)*

No special-purpose hardware should be required.

By special-purpose hardware are meant various graphics accelerators, 3D graphical boards, ...

Special-purpose hardware use *(General Constrain)*

Event Display may make use of special-purpose hardware if it is available.

By special-purpose hardware are meant various graphics accelerators, 3D graphics boards, ...

Online *(General Constrain)*

It should be possible to use Event Display in an on-line environment.

New graphical clients *(General System Property)*

It should be possible to use new graphical clients and environments.

A graphical client is the actual rendered which creates the picture on the screen. It is really Server in Client-Server terminology. New browsers will be available and it should be easy to use them.

Bibliography

[1] **The Large Electron Positron Collider**,
http://public.web.cern.ch/Public/ACCELERATORS/LepAcc.html

[2] **Aleph**,
http://alephwww.cern.ch/

[3] **CERN**,
http://cern.web.cern.ch/CERN/

[4] **The Large Hadron Collider**,
http://lhc.web.cern.ch/lhc/

[5] **The Compact Muon Solenoid**,
http://cmsinfo.cern.ch/Welcome.html

[6] **A Large Ion Collider Experiment**,
http://alice.web.cern.ch/Alice/

[7] **The Large Hadron Collider Beauty Experiment**,
http://lhcb.web.cern.ch/lhcb/

[8] **Tevatron**,
http://www-bd.fnal.gov/tevatron/

[9] **Delphi**,
http://delphi.web.cern.ch/Delphi/

[10] **BaBar**,
http://www.slac.stanford.edu/BFROOT/

[11] **Chorus**,
http://choruswww.cern.ch/

[12] **ATLAS Technical Proposal**, CERN/LHCC/94-43, LHCC/P2,
http://atlasinfo.cern.ch/ATLAS/TP/tp.html

[13] **ATLAS Proposal and Technical Design Reports**, http://atlasinfo.cern.ch/ATLAS/internal/tdr.html

[14] **ATLAS Detector and Physics Performance Technical Design Report**, CERN/LHCC/99-15, http://atlasinfo.cern.ch/Atlas/GROUPS/PHYSICS/TDR/access.html

[15] Brian W.Kernighan, Dennis M.Ritchie: **The C Programming Language**, ISBN:0-13-110362-8

[16] **Object-Oriented Programming and the Objective-C Language**, NeXT Computer, Inc., 1993, http://www.toodarkpark.org/computers/objc/

[17] **Subject-Oriented programming**, http://www.research.ibm.com/sop/

[18] **Aspect-Oriented Programming**, http://www.parc.xerox.com/csl/projects/aop/

[19] **AspectJ**, http://aspectj.org/servlets/AJSite

[20] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, **Design Patterns, Elements of Reusable Object-Oriented Software**, ISBN:0-201-63361-2, http://hillside.net/patterns/patterns.html

[21] David R.Musser, Atul Saini, **STL Tutorial and Reference Guide, C++ Programming with the Standard Template Library**

[22] P.P. Chen, **The entity-relationship model: toward a unified view of data**, ACM Transactions on Database Systems, 1:1 pp 9-36, 1976

[23] **The Class Relation Methodology**, Phillipe Desfray, **Object Oriented Modelling**, Addison Wessley,1994

[24] James Rumbaugh, **Object Oriented Modelling and Design**, ISBN:0-13-630054-5

[25] Grady Booch, **Object Oriented Modelling and Design with Applications**, ISBN:0-8053-5340-2

[26] **The Rational Unified Process**,
http://www.rational.com/products/rup/index.jsp

[27] **The Extreme Programming**,
http://www.xProgramming.com/,
http://www.extremeprogramming.org/

[28] **The Feature Driven Programming**,
Coad, De Luca, Lefebrve, **Java Modeling in Color with UML**, Prentice Hall 1999

[29] **The Cathedral and the Bazaar**,
http://www.tuxedo.org/ esr/writings/cathedral-bazaar/

[30] **The Open Source**,
http://www.opensource.org

[31] **GNU's Not Unix!**,
http://www.gnu.org/

[32] **Linux**,
http://www.linux.org/

[33] **The Capability Maturity Model**,
http://www.sei.cmu.edu/cmm/cmms/cmms.html

[34] The Carnegie Mellon Software Engineering Institute,
http://www.sei.cmu.edu/

[35] **The Rational Rose**,
http://www.rational.com/products/rose/index.jsp

[36] **Persistence of Vision**,
http://www.povray.org/

[37] **The Software Through Pictures**,
http://www.aonix.co.uk/

[38] **Together**,
http://www.oi.com/index.html

[39] **ArgoUML**,
http://argouml.tigris.org/

[40] Bernard Meyer, **Eiffel, The Language**, ISBN:0-13-247925-7,
Bernard Meyer, **Object-Oriented Software Construction**, ISBN:0-13-629155-4,
http://www.eiffel.com/

[41] Bjarne Stroustrup, **The C++ Programming Language**, ISBN:0-201-88954-4,
John J.Barton, Lee R.Nackman, **Scientific and Engineering C++ - An introduction with Advanced Techniques and Examples**, ISBN:0-201-53393-6,

[42] **Smalltalk**,
http://www.smalltalk.org/

[43] **Fortran**,
http://www.fortran.com

[44] **Perl**,
http://www.perl.org/

[45] **Python**,
http://www.python.org/

[46] **Tcl/Tk**,
http://www.scriptics.com/

[47] **The Extensible Markup Language**,
http://www.w3.org/XML/

[48] **The Virtual Reality Modeling Language**,
http://www.web3d.org/TaskGroups/x3d/specification/

[49] **The Extensible 3D**,
http://www.web3d.org/x3d.html

[50] James Gosling, **Java, An Overview**,
http://java.sun.com/people/jag/OriginalJavaWhitepaper.pdf,
James Gosling, Bill Joy, Guy L., Jr. Steele, **The Java Language Specification**, ISBN:0201634511,
http://java.sun.com/docs/books/jls/,
http://java.sun.com

[51] **MORTRAN**,
http://www.slac.stanford.edu/exp/e871/documentation/offline/mortran.html

[52] **AFS**,
http://www.openafs.org/

[53] **The Concurrent Versions System**,
http://www.loria.fr/ molli/cvs/doc/cvs_toc.html

[54] **Java3D**,
http://java.sun.com/products/java-media/3D/index.html

[55] **Model-View-Controller Architecture**,
http://www.jdl.co.uk/files/MVC.pdf

[56] **The CERN Programm Library for Fortran77**,
http://wwwinfo.cern.ch/asd/cernlib/

[57] **Geant4**,
http://wwwinfo.cern.ch/asd/geant4/geant4.html

[58] **Geant3**,
http://wwwinfo.cern.ch/asd/geant/index.html

[59] **Pythia6**,
http://shakespeare.thep.lu.se/tf2/staff/torbjorn/Pythia.html,
Pythia7,
http://shakespeare.thep.lu.se/ leif/Pythia7/Welcome.html

[60] **ZEBRA**,
http://wwwinfo.cern.ch/asdoc/zebra_html3/zebramain.html

[61] **The XML Format For Aida Histograms**,
http://www.slac.stanford.edu/ pspence/aidaproposal.htm

[62] **CMZ**,
http://wwwinfo.cern.ch/cmz/

[63] **Gaudi**,
http://lhcb-comp.web.cern.ch/lhcb-comp/Components/html/GaudiMain.html

[64] **The ATLAS Generic Detector Description**,
http://web1.cern.ch/Atlas/GROUPS/DATABASE/detector_description/,

[65] **SLUG**,
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/DOCUMENTS-
/SLUG_DOC/slug.html

[66] **DICE**,
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/DOCUMENTS-/DICE_320/dice320.html

[67] **ATRECON**,
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/DOCUMENTS-/reconstruction.html

[68] **Age**,
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/DOCUMENTS-/DICE95/AGE.html,
C.Arnault, J.Hřivnáč, et.al, **A Generic Approch to the Detector Description in ATLAS**, CHEP Conference in Padova, 2000

[69] **ATLSIM**,
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/DOCUMENTS-/simulation.html#atlsim

[70] **ATRIG**,
http://atlasinfo.cern.ch/Atlas/GROUPS/DAQTRIG/ATRIG/

[71] **The ATLAS Reconstruction and Visualisation Eneivronment**,
http://burnett.home.cern.ch/burnett/arve/

[72] **Provisional Analysis Skeleton for OO-ATLAS**,
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/OO/applications/Paso/

[73] **The Architecture TaskForce**,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/atf/

[74] Lassi Tuura, **Software Release Tools**,
http://atlasinfo.cern.ch/Atlas/GROUPS/SOFTWARE/DOCUMENTS-/srt_html/index.html

[75] **The Quality Control Group**,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/qc/

[76] **Athena**,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/

[77] **The Architecture Review Committee**,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/architecture/

[78] **The ATLAS Software Process**,
http://atlas.web.cern.ch/Atlas/GROUPS/SOFTWARE/OO/asp/

[79] **Text Scene**,
http://atlasinfo.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/Graphics-/JavaDoc/net/hep/atlas/Graphics/Text/package-summary.html

[80] **XML Scene**,
http://atlasinfo.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/Graphics-/JavaDoc/net/hep/atlas/Graphics/XML/package-summary.html

[81] **Aravis**,
http://atlas.web.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/Aravis/

[82] **Atlantis**,
http://atlasinfo.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/Atlantis/

[83] **WIRED**,
http://atlas.web.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/WIRED/, J.Hřivnáč, **WIRED (& Atlas)**, 3rd LHC Computing Workshop in Marseilles, 1999

[84] **GraXML**,
http://atlas.web.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/GraXML/

[85] **Object Browser**,
http://atlasinfo.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/Graphics-/JavaDoc/net/hep/atlas/Graphics/ObjectBrowser/package-summary.html

[86] **The Java Analysis Studio**,
http://atlas.web.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/JAS/

[87] **AIDA Scene**,
http://atlasinfo.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/Graphics-/JavaDoc/net/hep/atlas/Graphics/AIDA/package-summary.html

[88] **The Abstract Interfaces for Data Analysis**,
http://wwwinfo.cern.ch/asd/lhc++/theProject/index.html

[89] **Test Scene**,
http://atlasinfo.cern.ch/Atlas/GROUPS/GRAPHICS/Texts/Documentation/Graphics-/JavaDoc/net/hep/atlas/Graphics/Test/package-summary.html

[90] **FreeHEP**,
http://java.freehep.org/lib/freehep/doc/web/index.html

[91] M.Smižanská, J.Hřivnáč, $\mathrm{B_d^0}$ B $\rightarrow$ J/ψ K^0 μ X **reaction: rates, dilution factor and the error on** $\sin(2\phi)$ **as a function of trigger** p_t **threshold**, ATLAS Internal Note, PHYS-NO-022,1993

[92] **Feasibility of beauty baryon polarization measurement in** Λ^0 **J/**ψ **decay channel by ATLAS-LHC**
, by J.Hřivnáč, R.Lednický, M.Smižanská; (12 p) . 1994 . Publ. Ref: J. Phys. G: 21 (1995) 629-683

[93] P.Eerola, **Measurement of CP-violation in B-decays with the ATLAS**, ATLAS Internal Note, PHYS-NO-009,1992

[94] M.V.Danilov, **Heavy flavour physics**, Proceedings of the Lepton-Photon Symposium & and Europhysics Conference on High Energy Physics, Geneva, 1991,vol.2,p.333

[95] M.Bolto et al., **Further comments about B-physics in pp interactions**, SSCL-538/A, SLAC-PUB-5795,1992

[96] K.Heller, Proceedings of the VII-th Int.Symp. on High Energy Spin Physics, Protvino, 1986vol.I,p.81.

[97] L.Pondrom, Phys.Rep. 122(19985) 57.

[98] B.Andersson et al., Phys.Lett. 85B(1979) 417.

[99] T.A.De Grand, H.I.Miettinen, Phys.Rev. D24(1981) 2419.

[100] B.V.Struminsky, Yad.Fiz.34(1981) 1954.

[101] R.Lednicky, Czech.J.Phys.B33(1983) 1177, Z.Phys.C26(1985) 531.

[102] P.M.Ho et al., Phys.Rev.Lett. 65 (1990) 1713.

[103] A.Morelos et al., FERMILAB-Pub-93/167-E.

[104] R.Lednicky, Yad.Fiz.43(1986)1275 (Sov.J.Nucl.Phys. 43(1986), 817).

[105] G.Kane, Y.P.Yao, Nucl.Phys.B137(1978) 313.

[106] J.Szwed, Phys.Lett.105B(1981) 403.

[107] W.G.D. Dharmaratna, Gary R. Goldstein, Phys.Rev.D41(1990)1731.

[108] B.V.Batyunya et al., Czech.J.Phys.B31(1981) 11.

[109] C.M.Troshin, H.E.Tyurin, Yad.Fiz.38(1983) 1065.

[110] A.V.Efremov, O.V.Teryaev, Phys.Lett.B150(1985) 383.

[111] A.N.Aleev et al., Yad.Fiz.43(1986) 619.

[112] P.Chauvatet et al., Phys.Lett. 199B(1987) 304.

[113] The ATLAS Collaboration, CERN/LHCC/93-53, Oct.1993.

[114] W.Hoffmann, DESY 93-026(1993).

[115] H.Albrecht et al., DESY 93-156(1993).

[116] A.H.Ball et al., J.Phys. G: Nucl. Part. Phys.18(1992) 1703.

[117] UA1 Collaboration, Phys.Lett. 273B(1991) 544.

[118] CDF Collaboration, Phys.Rev. D47(1993) R2639.

[119] I.Gavrilenko, ATLAS Internal Note INDET-NO-016,1992.

[120] A.F.Falk and M.E.Peskin, SLAC-PUB-6311,1993.

[121] R.Lednicky, DrSc Thesis, JINR-Dubna 1990, p.174 (in russian).

List of Figures

Index