

# ALICE Collaboration

---



# Contents

---

<b>2</b>	<b>Overview of the computing framework</b>	<b>2</b>
2.1	The development of AliRoot . . . . .	2
2.2	ROOT framework . . . . .	2
2.3	AliRoot framework . . . . .	4
2.3.1	The function of AliRoot . . . . .	4
2.3.2	Principles of AliRoot design . . . . .	5
2.3.3	Data structure design . . . . .	6
2.3.4	Offline detector alignment and calibration model . . . . .	7
2.3.5	Tag database . . . . .	10
2.3.6	LHC common software . . . . .	10
2.4	Software Development Environment . . . . .	11
2.5	Maintenance and distribution of AliRoot . . . . .	12
2.5.1	Code development tools . . . . .	13
	<b>References</b>	<b>15</b>



## 2 Overview of the computing framework

---

The objective of the ALICE computing framework is twofold: the simulation of the primary pp and heavy-ion interactions and of the resulting detector response; and the reconstruction and analysis of the data coming from simulated and real interactions.

When building the ALICE detector, the optimization of the hardware design and the preparation of the code and computing infrastructure require a reliable simulation and reconstruction chain implemented by a distributed computing framework. Since few years, ALICE has been developing its computing framework called AliRoot [1]. This has been used to perform simulation studies for the Technical Design Reports of all ALICE sub-detectors, in order to optimize their design. It has also been used for the studies of the ALICE Physics Performance Report to assess the physics capabilities of the ALICE detector and for the Computing and Physics Data Challenges. A representation of the ALICE detector geometry within the AliRoot simulation framework is illustrated in Colour Figure I. This chapter describes the development and present structure of the AliRoot framework.

### 2.1 The development of AliRoot

The development of the ALICE computing framework started in 1998. At that time, the size and complexity of the LHC computing environment had already been recognized. Since the early 1990s it had been clear that the new generation of programs would be based on Object-Oriented (OO) techniques [2]. A number of projects were started, to replace the FORTRAN CERNLIB [3], including PAW [4] and GEANT 3 [5] with OO products.

In ALICE, the computing team developing the framework and the physicists developing physics software and algorithms are in a single group. This organization has proved effective in improving communication, and in encouraging the software developers to constantly provide working tools while progressing towards the final computing system. Thanks to this close collaboration, the ALICE physics community supported a rapid and complete conversion to OO/C++, under the condition that a working environment at least as good as GEANT 3, CERNLIB and PAW could be provided quickly.

After a short but intense evaluation period, the ALICE computing team concluded that one such framework existed, namely the ROOT system [6], which is now the de facto standard of HEP software and a major foundation of the software of the LHC Computing Grid (LCG) [7] project. The decision to adopt the ROOT framework was therefore taken and the development of the ALICE computing framework, AliRoot, started immediately, making full use of all the ROOT potential.

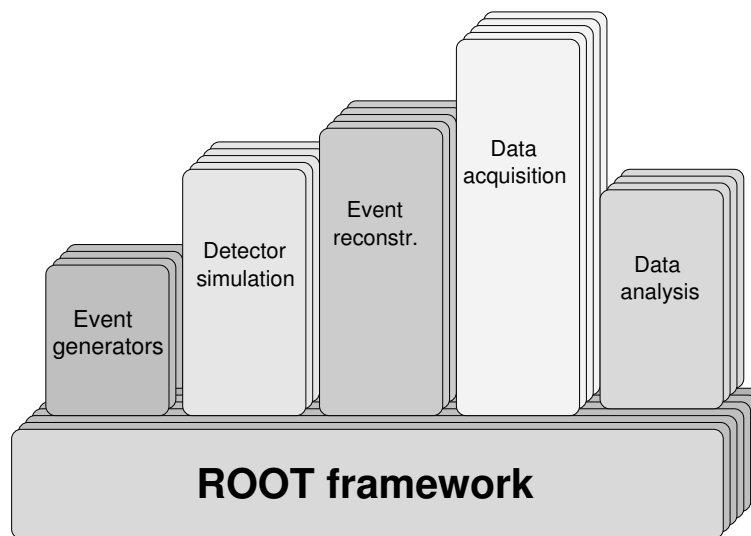
This process has resulted in a tightly knit computing team with one single line of development. The move to OO was completed successfully resulting in one single framework, entirely written in C++ and soon adopted by the ALICE users. The detector Technical Design Reports and the Physics Performance Report were written using simulation studies performed with this framework, while the system was in continuous development. This choice allowed the ALICE developers to address both the immediate needs and the long-term goals of the experiment with the same framework as it evolved into the final one, with the support and participation of all ALICE users.

### 2.2 ROOT framework

The ROOT framework<sup>1</sup>, schematically shown in Fig. 2.1, provides a set of interacting classes and an environment for the development of software packages for event generation, detector simulation, event

---

<sup>1</sup>In HEP, a framework is a set of software tools that enables data processing. For example CERNLIB was a toolkit to build a framework. PAW was the first example of integration of tools into a coherent ensemble specifically dedicated to data analysis. ROOT is a new-generation, Object-Oriented framework for large-scale data-handling applications.



**Figure 2.1:** The ROOT framework and its application to HEP.

reconstruction, data acquisition, and a complete data analysis framework including all PAW features. An essential component of ROOT is the I/O subsystem that allows one to store and retrieve C++ objects and is optimized for efficient access to very large quantities of data.

ROOT has evolved over the years to become a mature and complete framework, embodying a very large spectrum of features. It is outside the scope of this document to provide a full description of ROOT. In the following paragraphs we shall limit ourselves to outlining the major features that are relevant for the ALICE computing framework.

ROOT is written in C++ and offers integrated I/O with class schema evolution, an efficient hierarchical object store with a complete set of object containers, a C++ interpreter allowing one to use C++ as scripting language, advanced statistical analysis tools (multidimensional histogramming, several commonly used mathematical functions, random number generators, multi-parametric fit, minimization procedures, cluster finding algorithms etc.), hyperized documentation tools, geometrical modelling, and advanced visualization tools. The user interacts with ROOT via a graphical user interface, the command line or script files in C++, which can be either interpreted or dynamically compiled and linked.

ROOT presents a coherent and well integrated set of classes that easily inter-operate via an object bus provided by the interpreter dictionaries (these provide extensive Run Time Type Information, RTTI, of each object active in the system). This makes ROOT an ideal basic infrastructure on which an experiment's complete data handling chain can be built: from DAQ, using the client/server and shared memory classes, to database, distributed analysis, thanks to the PROOF facility, and data presentation.

The Parallel ROOT Facility [8], PROOF, makes use of the inherent parallelism in event data and implements an architecture that optimizes I/O and CPU utilization in heterogeneous clusters with distributed storage. Being part of the ROOT framework, PROOF inherits the benefits of a well-performing object storage system and a wealth of statistical and visualization tools. Queries are automatically parallelized and balanced by the system which implements a simple but very effective master-worker model. The results of the queries are joined together by the system and presented to the users.

The backbone of the ROOT architecture is a layered class hierarchy organized in a single-rooted class library where classes inherit from a common base class `TObject`. This has proved to be well suited for our needs (and indeed for almost all successful class libraries: Java, Smalltalk, MFC, etc.). A ROOT-based program links explicitly with a few core libraries, and at run-time it loads dynamically additional libraries as needed, possibly via string activated class initializations (plugins).

One of the key components of ROOT is the CINT C/C++ interpreter. The ROOT system embeds CINT in order to be able to execute C++ scripts and C++ command line input. CINT also provides

ROOT with extensive RTTI capabilities covering essentially the totality of C++. The advantage of a C/C++ interpreter is that it allows for fast prototyping since it eliminates the typical time-consuming edit/compile/link cycle. Scripts can be compiled on-the-fly from the ROOT prompt with a standard C/C++ compiler for full machine performance.

ROOT offers a number of important elements that have been exploited in AliRoot as the basis for the successful migration of the users to OO/C++ programming.

The ROOT system can be seamlessly extended with user classes that become effectively part of the system. The corresponding libraries are loaded dynamically and the user classes share the same services of the native ROOT classes, including object browsing, I/O, dictionary and so on.

ROOT can be driven by scripts having access to the full functionality of the classes. For production, it eliminates the need for configuration files in special formats, since the parameters can be entered via the setters of the classes to be initialized. This is also particularly effective for fast prototyping. Developers can implement code working with a script within the same ROOT interactive session. When the code is validated, it can be compiled and made available via shared libraries, and the development can restart via scripts. This has been observed to lower the threshold considerably for new C++ users and has been one of the major enabling factors in the migration of users to the OO/C++ environment.

The ROOT system has been ported to all known Unix variants (including many different C++ compilers), to Windows from 9x to XP, and to Mac OS X.

ROOT is widely used in particle and nuclear physics, notably in all major US labs (FermiLab, Brookhaven, SLAC), and most European labs (CERN, DESY, GSI). Although initially developed in the context of particle and nuclear physics, it can be equally well used in other fields where large amounts of data need to be processed, such as astronomy, biology, genetics, finance, insurance, pharmaceutical, etc.

ROOT is the foundation of the LCG software, providing persistency for LHC data and data analysis capabilities.

The ROOT system has recently been interfaced with emerging Grid middleware in general. A first instantiation of this interface was with the ALICE-developed AliEn system [9]. In conjunction with the PROOF system, this has allowed the realization and demonstration of a distributed parallel computing system for large-scale production and analysis.

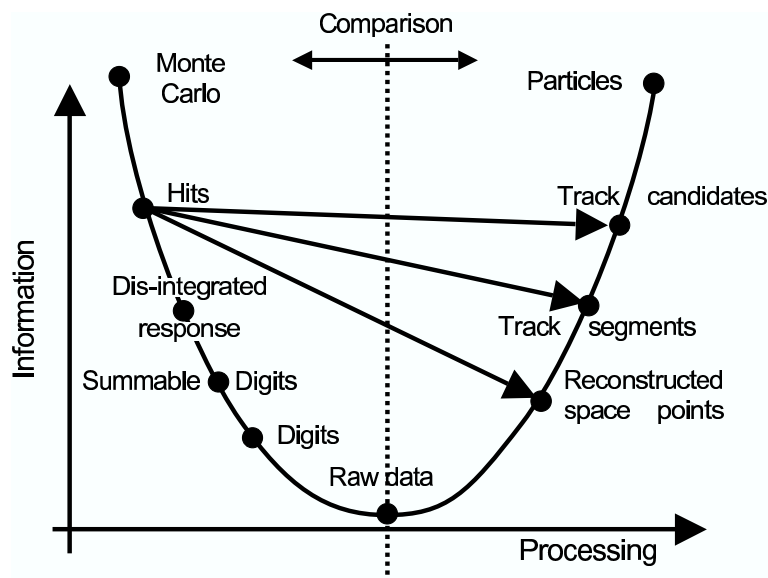
This interface is being extended to other middleware systems as these become available.

## 2.3 AliRoot framework

### 2.3.1 The function of AliRoot

The functionality of the AliRoot framework is shown schematically in Fig. 2.2. Simulated data are generated via Monte Carlo event generators. The generated tracks are then transported through the detector via detector simulation packages such as GEANT 3, FLUKA [10] and GEANT 4 [11]. These packages generate a detailed energy deposition in the detector, which is usually called a ‘hit’, with a terminology inherited from GEANT 3. Hits are then transformed into an ideal detector response, which is then transformed into the real detector response, taking into account the electronic manipulation of the signals, including digitization. As explained in Chapter ?? on simulation, the need to ‘superimpose’ different simulated events has led us to develop an intermediate output format called ‘summable digits’. These are high-resolution, zero-threshold digits, which can be summed when different simulated events are superimposed. Digits are then transformed into the format that will be output by the electronics of the detectors, called ‘raw’ format. From here on the processing of real or simulated data is indistinguishable.

The data produced by the event generators contain the full information about the generated particles: Particle Identification (PID) and momentum. As these events are processed via the simulation chain, the information is disintegrated and reduced to that generated by particles when crossing a detector. The reconstruction algorithms reconstruct the information about the particle trajectory and identity from the information contained in the raw data. In order to evaluate the software and detector performance,



**Figure 2.2:** Data processing framework.

simulated events are processed through the whole cycle and finally the reconstructed information about particles is compared with the information taken directly from the Monte Carlo generation.

Fast simulations are ‘shortcuts’ in the whole chain, as indicated by the arrows in the figure. These increase the speed of the simulation at the expense of the details of the result, and are used for special studies. The AliRoot framework implements several fast simulation algorithms.

The user can intervene in this framework-driven cycle to implement private code provided it respects the interfaces exposed by the framework. I/O and user interfaces are part of the framework, as are data visualization and analysis tools and all procedures and services of general interest. The scope of the framework evolves with time following the needs and understanding of the users.

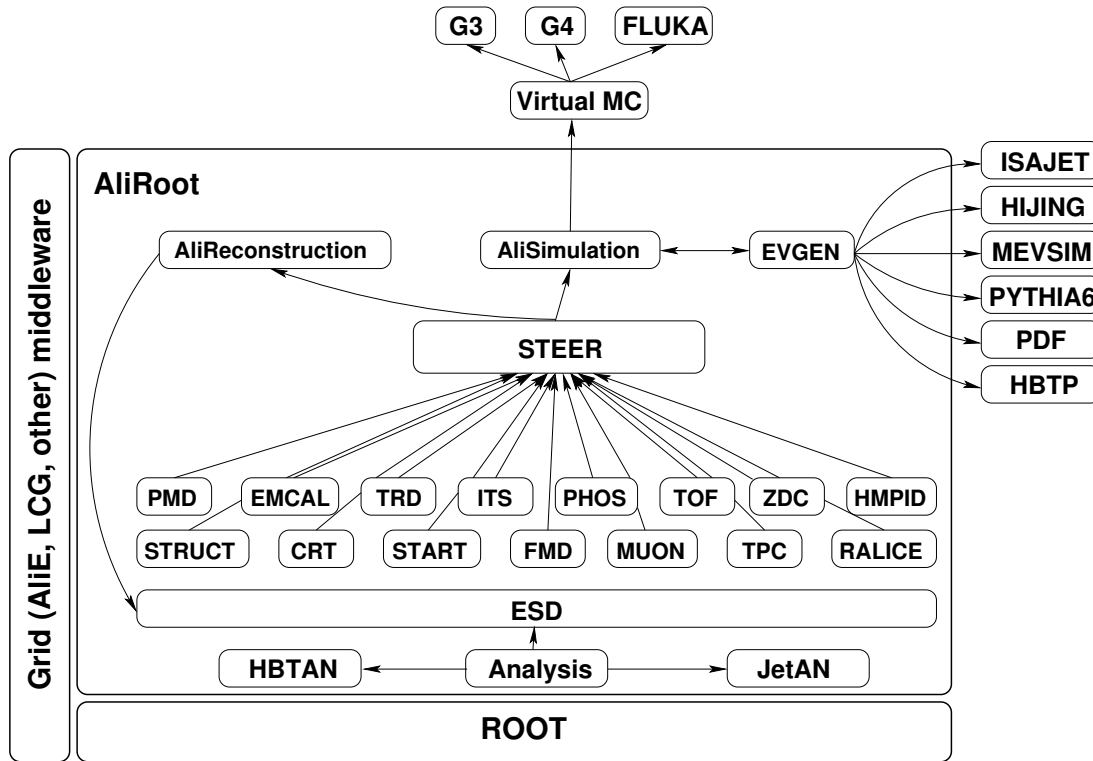
### 2.3.2 Principles of AliRoot design

The basic principles that have guided us in the design of the AliRoot framework are re-usability and modularity, with the objective of minimizing the amount of unused or rewritten user code and maximizing the participation of the physicists in the development of the code.

**Modularity** allows the replacement of parts of the system with minimal or no impact on the rest. Not every part of the system is expected to be replaced and modularity is targeted at those elements that could potentially be changed. There are elements that we do not plan to change, but rather to develop in collaboration with their authors. Whenever an element has to be modular in the sense above, we define an abstract interface to it. Some major examples are:

- Different transport Monte Carlo’s can be used without change in the scoring or geometry description codes for the different sub-detectors. This is further described in detail in Chapter ???. This is realized via a set of virtual interfaces that are part of the ROOT system, called Virtual Monte Carlo.
- Different event generators can be used and even combined via a single abstract interface.
- Different reconstruction algorithms for each sub-detector can be used with no effect on the rest of the code. This includes also the algorithms being developed for HLT, to allow their comparison with the offline ones. Again this is implemented via abstract interfaces.

The codes from the different detectors are independent so that different detector groups can work concurrently on the system while minimizing interference.



**Figure 2.3:** Schematic view of the AliRoot framework.

**Re-usability** is the protection of the investment made by the physicist programmers of ALICE. The code contains a large amount of scientific knowledge and experience and is thus a precious resource. We preserve this investment by designing a modular system in the sense above and by making sure that we maintain the maximum amount of backward compatibility while evolving our system.

The AliRoot framework is schematically shown in Fig. 2.3. The central module is STEER and it provides several common functions such as:

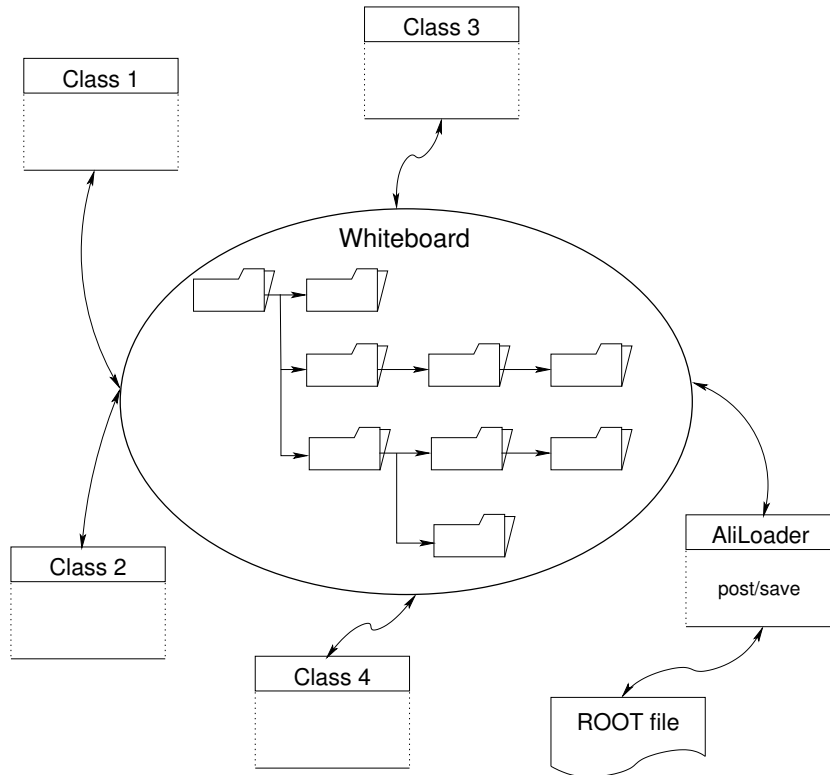
- steering of program execution for simulation, reconstruction and analysis;
- general run management, creation and destruction of data structures, initialization and termination of program phases;
- base classes for simulation, event generation, reconstruction, detector elements.

The sub-detectors are independent modules that contain the specific code for simulation and reconstruction while the analysis code is progressively added. Detector response simulation can be performed via different transport codes via the Virtual Monte Carlo mechanism [12].

The same technique is used to access different event generators. The event generators are accessed via a virtual interface that allows the loading of different generators at run time. Most of the generators are implemented in FORTRAN but the combination of dynamically loadable libraries and C++ ‘wrapper’ classes makes this completely transparent to the users.

### 2.3.3 Data structure design

Object-Oriented design is based on data encapsulation. Data processing in HEP is based on successive passes on data that is subsequently transformed and analysed. It has been noted that in this case data encapsulation may lead to tangled relationships between classes, introducing unwanted dependencies



**Figure 2.4:** AliRoot whiteboard.

and making the design difficult to evolve. To avoid this problem and still maintain the benefits of Object-Oriented design, we have decided to exploit the ROOT folder facility to create a data ‘whiteboard’ as shown in Figure 2.4.

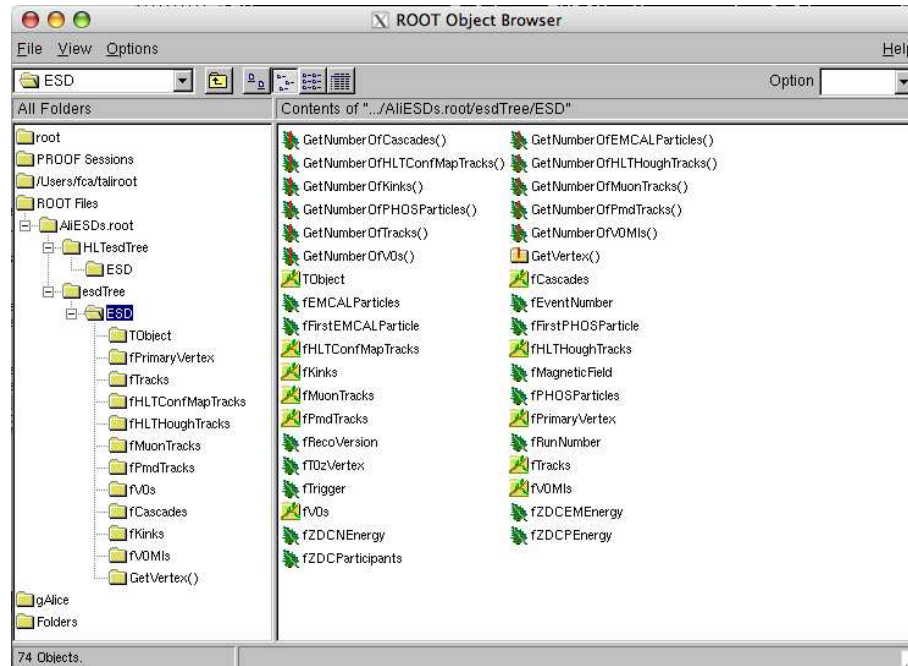
The main idea of the data whiteboard is that I/O is delegated to a set of service classes that read data from the files and ‘post’ them to a set of ROOT folders with the same semantics as a Unix file system. The same classes perform the opposite operation and ‘unload’ the data into a file. This has had the double advantage of concentrating I/O operations in a single set of classes, and simplifying data dependencies between modules, which therefore remain independent of each other. Figure 2.5 presents the AliRoot folder structure for the ESD.

### 2.3.4 Offline detector alignment and calibration model

The ALICE offline calibration and alignment framework will provide the infrastructure for the storage and the access to the experiment condition data. These include most non-event data and notably the calibration and alignment information used during reconstruction and analysis. Its design is primarily driven by the necessity for a seamless access to a coherent set of calibration and alignment information in a model where data and processing are distributed worldwide over many independent computing centres.

The condition data are contained in ROOT objects stored into read-only ROOT files and registered in the ALICE file catalogue. Evolution will be handled through versioning, thus avoiding the necessity to develop a complicated internal synchronization schema. The file catalogue will associate the metadata describing the condition information with the logical files where the information is stored. These tags will be the primary method for file access during the production and analysis phase.

The usage of the ROOT I/O object store capabilities together with the navigational, metadata and distributed access capabilities of the ALICE Grid file catalogue avoids the development of a more complicated but functionally equivalent structure based on distributed relational database systems.



**Figure 2.5:** Example of AliRoot ESD folders.

The framework provides a set of classes to access and manipulate the condition objects. These can be stored in a distributed, Grid-enabled environment or copied onto the local disk of a machine potentially disconnected from the network. In this case the metadata will be encoded in the file name. The classes are designed so that, once the access method is specified, no other change in the code is necessary.

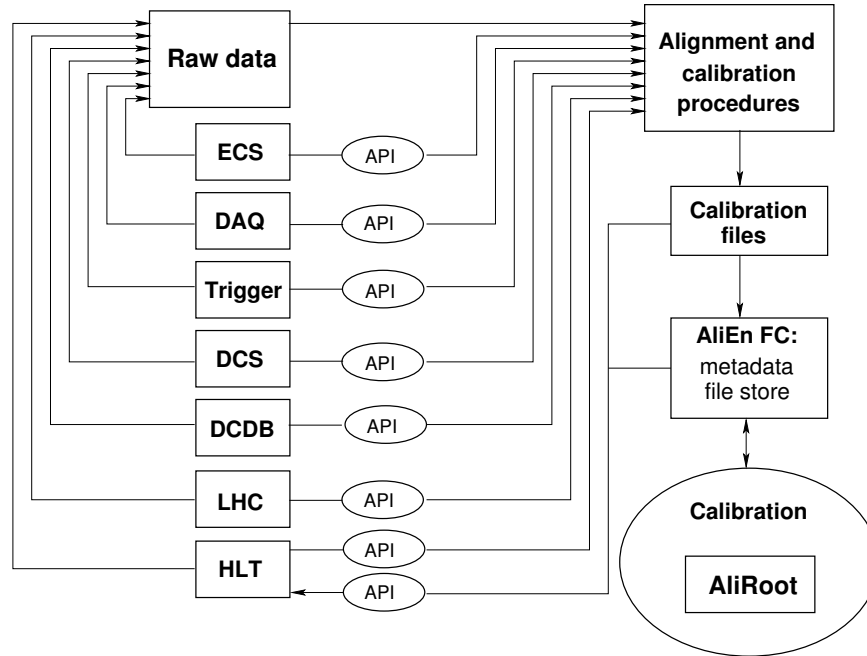
The source data for condition information, both static and dynamic will be stored in on-line and archive databases, which are populated during the detector construction and integration phase and the data taking periods. These databases include:

- Detector Construction Database (DCDB): Used by individual sub-detector groups during the production and integration phase and containing static information on detector elements performance, localization and identification.
- Experiment Control System (ECS) database: Contains information on the active sub-detector partition during data taking and the function of this partition.
- Data Acquisition (DAQ) database: A repository for data acquisition related parameters and for resource assignments to data acquisition tasks: current and stored configurations, current and stored run parameters.
- Trigger database: Contains the ALICE trigger classes (including the input to the Central Trigger Processor), and the definition of the trigger masks.
- Detector Control System (DCS) databases: Configuration DB, containing the configuration parameters for systems and devices (modules and channels), and the front-end configuration (busses and thresholds); Archive DB containing the monitored detectors and device parameters.
- High-Level Trigger (HLT) database: A TAG/ESD database containing HLT information relevant for physics studies and offline event selection.
- LHC Machine database: Machine status and beam parameters.

The alignment and calibration framework will collect the information from the various sources and make it accessible for offline distributed processing, either via periodic polling or asynchronous access. The information will be stored in ROOT files published in the ALICE Grid catalogue and annotated with the proper condition tags in the form of file metadata.

Most of the data flow is uni-directional toward the offline with the exception of the HLT system, which requires access to the condition data but may also generate them.

The relation of the alignment and calibration procedures and the various sources of condition data are presented in Fig. 2.6.



**Figure 2.6:** Schematic view of the relations of the calibration and alignment procedures to the condition data sources and the AliRoot and AliEn frameworks.

The update and access frequency to the condition objects will be once per run. The objects, however, can contain more finely-grained information depending on time, event number or other variables, for example in the form of a histogram. Each sub-detector will have its own condition storage partition, with possibly a different granularity and condition parametrization within the objects. There will be partitions containing information common for all ALICE sub-detectors, for example magnetic field maps and the LHC machine information. This design has been developed and verified by collecting and analyzing an extensive set of user requirements and use cases.

The sub-detectors and global alignment framework will be based on the ROOT geometrical modeller package [13]. The common alignment framework will provide facilities for the retrieval of the right alignment files and for the ‘correction’ of the position of the detector elements with the alignment data. The calibration algorithms are intrinsically detector-specific and therefore their development is under the responsibility of the sub-detector groups.

At the time of publication of this document, work is going on in parallel on the development of the framework access classes, the API for gathering information from the various sources of condition data and detector-specific alignment and calibration code. These will be tested during the distributed Physics Data Challenges in 2005 and 2006 and will be ready for the start of data taking in April 2007.

### 2.3.5 Tag database

The ALICE experiment is expected to produce many petabytes of data per year. Most user analyses are conducted on subsets of events. Typically, the events are organized into files stored on mass storage systems. The performance of analysis jobs strongly depends on file management functions such as finding what files contain the wanted events, locating the files, transferring the files to a convenient location for analysis and removing the files afterwards, or moving the analysis task to the file location.

ALICE is implementing a system that will reduce the time needed to perform an analysis by providing to each analysis code only the events of interest as defined by the users themselves. This task will be performed first of all by imposing event selection criteria within the analysis code and then by interacting with software that is designed to provide a file-transparent event access for analysis programs. The latter is derived from a product developed and used by the STAR [14] Collaboration.

This software, called Grid Collector (GC) [15], is designed to provide file-transparent event access for analysis programs. This software allows users to specify their requests for events as sets of conditions on physically meaningful attributes, such as triggers, production versions and other tags. The GC resolves these conditions into a list of relevant events and a list of files containing the events. It is able to locate the files and transfer the files as needed. Its main components are the Query Interpreter and the Event Iterator.

- A component called Bitmap Index [16] takes as input the values of selected attributes and generates indices pointing to the events.
- The Query Iterator takes the selection criteria provided by the users and translates them with the help of the produced indices into a list of files and events in these files that satisfy the selection.
- The Event Iterator interfaces the analysis framework to the GC. It retrieves the selected events and passes them to the analysis code.

The current prototype of the TAG database stores information about several stages of data processing such as: LHC machine, run and detector information and a set of event features selected by the Physics Working Groups for fast selection of an event sample from the full data set. All information will be stored in ROOT files. Two scenarios are considered for their creation:

- ‘On-the-fly’ creation. The TAG files are created by the reconstruction code and registered in the ALICE file catalogue together with the ESDs.
- ‘Post-processing’. After the produced ESDs are registered, a process will loop over all the registered files of each run in order to create the tag ROOT files and register them in the ALICE file catalogue.

We will use the GC Bitmap Index to produce the indices for every attribute stored in these TAG files. The user uses the GC’s Query Interpreter to have fast and efficient access to the events of interest via a set of selection criteria on TAG attributes.

### 2.3.6 LHC common software

The LHC Computing Grid (LCG) was launched in March 2002 following the recommendations of the LHC Computing Review [17]. Its objective is to provide the LHC experiments with the necessary computing infrastructure to analyse their data. This should be achieved via the realization of common projects in the field of application software and Grid computing, deployed on a common distributed infrastructure. The project is divided into two phases, one phase of preparation and prototyping (2002–2005), and one of commissioning and exploitation of the computing infrastructure (2006–2008).

An overview Software and Computing Committee (SC2) has organized Requirement Technical Assessment Groups (RTAGs) defining requirements. These are received by the Project Execution Board that organizes the project activity in several workpackages.

ALICE has been very active in the setting up of this structure and ALICE members have served as chairs for several RTAGs. ALICE has elaborated a complete solution for handling the data, and intends to continue developing it while collaborating fully with the other experiments in the framework of the LCG project. ALICE welcomes sharing its software with the other experiments.

In the application area, the LCG project has decided to make ROOT one of the central elements of its development and ALICE fully supports this decision. ALICE sees its role in the LCG project as one of close collaboration with the ROOT team in developing base technologies (e.g. geometrical modeller, Virtual Monte Carlo) which are included directly in ROOT and made available to the other LHC experiments.

In the Grid technology area ALICE, successfully deployed and used its AliEn Grid framework. We are currently working in collaboration with LCG and the other experiments [18] to maximise the use of the common Grid infrastructure provided by LCG, interfacing the ALICE-specific AliEn services to it.

## 2.4 Software Development Environment

The ALICE software community is composed of small groups of two–three people who are geographically dispersed and who do not respond hierarchically to the Computing Coordinator. This affects not only the organization of the computing project but also the software development process itself as well as the tools that have to be employed to aid this development.

This situation is not specific to ALICE. It is similar to the other modern HEP experiments. However, this is a novel situation for HEP. In the previous generation of experiments, during the LEP era, although physicists from different institutions developed the software, they did a large share of the work while at CERN. The size of modern collaborations and software teams makes this model impractical. The experiments' computing programs have to be developed by truly distributed teams whose members meet very infrequently.

To cope with this situation, ALICE has elaborated a software process inspired by the most recent Software Engineering trends, in particular by the extreme programming principle [19]. In a nutshell, traditional software engineering aims at reducing the occurrence of change via a complete specification of the requirements and a detailed design before the start of development. After an attentive analysis of the conditions of software development in ALICE, we have concluded that this strategy cannot succeed. Therefore, we have adopted more flexible development methods and principles, which are outlined below:

- **Requirements are necessary to develop the code.** ALICE users express their requirements continuously with the feedback they provide. The computing core team redirects its priorities according to the user feedback. This way, we make sure that we are working on the highest priority item at every moment, thus maximizing the development efficiency. The developers meet at CERN three times per year during so-called 'offline weeks' that play a major role in reviewing the current state of the project, collecting user requirements and feedback, and planning of future activities.
- **Design is good for evolving the code.** AliRoot code is redesigned continuously since the feedback received from the users is folded directly into the design activity of the core computing team. At any given time there is a short-term plan to the next development phase, instead of the traditional long-term static objective.
- **Testing is important for quality and robustness.** While component testing is the responsibility of the groups providing modules to AliRoot, full integration testing is done nightly to make sure that the code is always functional. At present, the tests are concentrated on the main steps in

the simulation: event generation and transport, hits, detector response, summable digits, event merging, and digitization. In addition, the reconstruction is carried on including clusterization, reconstruction of tracks, vertices,  $V^0$  and cascades, particle identification and creation of ESD. The results of the tests are reported on the Web and are publicly accessible [20].

- **Integration is needed to ensure coherence.** The AliRoot code is collectively owned with a single code base handled via the CVS [21] concurrent development tool, where all developers store their code. The same CVS repository is used by the HLT project to store their algorithms, which are functionally integrated into the offline code. The AliRoot CVS contains both the production version and the latest (development) version. For every module, one or two developers can perform updates. The different modules are largely independent and therefore, even if a faulty modification is stored, the other modules still work and the global activity is not stopped. Having all the code in a single repository allows for a global overview to be easily carried out [22]. A hyperized code version is also extracted using the ROOT documentation tools [23].
- **Discussions are valuable for users and developers.** Discussion lists are commonplace in all computing projects. In ALICE, the discussion list is used to express requirements and evaluate design. Frequently, important design decisions are initiated by an e-mail discussion thread, in which all interested ALICE users can participate. Following such discussion, a redefinition of planning and sharing of work can also occur, thus adding flexibility to the project and allowing it to address both short-term and long-term needs.

The above development strategy is very close to the one used by successful Open-Source projects such as Linux, GNU, ROOT, KDE, GSL and so on, making it easy for us to interact with them.

## 2.5 Maintenance and distribution of AliRoot

The ALICE code is composed of one single OO framework implemented in C++ and with a multitude of users and developers participating in the design and implementation. Distributed development of such a large and rapidly evolving code is a challenging task which requires proper organization and tools.

The development model chosen by ALICE implies that any design becomes quickly obsolete since the development is driven by the most urgent needs of the user community. In this environment it becomes of particular importance to avoid the risk of an anarchic code development with the introduction of well-known design errors (see for instance [24]). To keep the code development under control, we hold regular code and design reviews and we profit from advanced software engineering tools developed in collaboration with computer science experts, see Section 2.5.1

For this model to work, a specific release policy has been elaborated during the first two years of existence of AliRoot. It is based on the principle of fixed release dates with flexible scope. Given that the priority list is dynamically rearranged, it is difficult to define the scope of a given release in advance. Instead, we decided to schedule the release cycle in advance. The current branch of the CVS repository is ‘tagged’ every week with one major release every six months or before a major production. The production branch of the CVS repository is tagged only for new check-ins due to bug fixes.

When a release is approaching, the date is kept fixed, and the scope of what is to be included is tailored. Large developments are moved to the current branch and only developments that can be completed within the time remaining are included. The flexible priority scheduling ensures that if a feature is urgent, enough resources are devoted to make it ready in the shortest possible time. As soon as it is ready, it is made available via a CVS tag on the active branch.

Special software tools are used to improve and verify the quality of the code. We rely on the ROOT memory checker [25] for fast detection of memory leaks. Extensive searching for runtime errors is done using the Valgrind tool [26]. The VTune profiling tool [27] is extremely helpful in the optimization of the code.

The installation process of AliRoot is specifically tailored to be efficient and reliable. AliRoot is one single package that links only to ROOT. Thanks to this minimal dependency, the installation does not require configuration management tools. A special attempt has been made to be independent from the specific version of the operating system and compiler. To ensure easy portability to any future platform, one of our coding rules states that the code must compile without warning on all supported platforms. At this time, these are Linux IA32 and IA64 architectures, DEC-Unix with True64, Solaris, and Mac OS X.

### 2.5.1 Code development tools

Very early in the development of AliRoot we decided to acquire tools to help us with code development and maintenance via a collaboration with computer science experts. We therefore established a collaboration with the IRST [28–30] at Trento, Italy, who were interested in developing state-of-the-art software engineering tools and to test them in production on a large code.

Distributed and collective code development requires a high degree of uniformity of the code. Developers come and go, and the code and its structure has to be readable and easily understandable. We realized this very early on and therefore we decided to adopt a small set of coding and programming rules [31]. It was soon clear that only an automatic tool could verify the compliance of the code with these rules. This tool has been developed in collaboration with IRST and is currently used to check the code for compliance. A table of the violations in all modules is published on the Web [32] every day.

Object-Oriented code design and development is best done using formal representation of the code structure, such as the one provided by the Unified Modelling Language (UML [33]). The problem with UML is that it is difficult to maintain consistency between the code design and the actual code implemented. To alleviate this problem, associated with the code checker there is a reverse-engineering tool that produces UML diagrams for all the AliRoot modules. It is run together with the nightly builds so that an up-to-date formal representation of the code design is always available. The results are published on the ALICE offline Web page [34]. This tool is essential in providing a constantly up-to-date design of the AliRoot code, which is used in code design discussions.

A new project has been established with IRST for the period 2004–2007. The main objectives are to produce a new set of software engineering tools:

- **Automated test case generation.** Automated test case generation is used for coverage testing. The implementation is based on genetic algorithms. All the test cases are considered as individuals of a population with chromosome-encoded test input values. Test cases are evolved by means of mutation (e.g., change value) and crossover (e.g., swap input value tails). The fittest individuals are the test cases that get closest to the target of test execution (for example, covering a given branch).
- **Introduction of Aspect-Oriented programming.** This paradigm permits the execution flow to be intercepted by an aspect in a well-defined point. Several usage scenarios will be investigated by the project, such as debugging, counting executions instances and timing of program sections and memory monitoring.
- **Code smell detection.** Code smell detection helps in the refactoring of the existing software. Refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code, while it improves its internal structure. It helps in introducing a disciplined way to ‘clean-up’ the code, while improving the code design during development and reducing the risks associated with a fast development of the code. The tool will reveal the most common design mistakes, called ‘smells’ so that they can be analysed and reduced.



# References

---

## Chapter 2

- [1] <http://www.cern.ch/ALICE/Projects/offline/aliroot/Welcome.html>.
- [2] See for instance G. Booch, *Object-oriented Analysis and Design with Applications, 2nd edition.*, (Benjamin Cummings, Redwood City, 1993) ISBN 0-8053-5340-2.
- [3] <http://wwwinfo.cern.ch/asd/cernlib>.
- [4] <http://wwwinfo.cern.ch/asd/paw>.
- [5] R. Brun, F. Bruyant, M. Maire, A.C. McPherson, P. Zanmarini, GEANT3 User Guide, CERN Data Handling Division DD/EE/84-1 (1985);  
<http://wwwinfo.cern.ch/asdoc/geantold/GEANTMAIN.html>.
- [6] <http://root.cern.ch>.
- [7] <http://lcg.web.cern.ch/LCG>.
- [8] M. Ballintijn, R. Brun, F. Rademakers and G. Roland, Distributed parallel analysis framework with PROOF, Proc. of TUCT004.
- [9] P. Saiz *et al.*, Nucl. Instrum. Methods **A502** (2003) 437–440;  
<http://alien.cern.ch/>.
- [10] A. Fassò *et al.*, in *Proc. of Computing in High Energy and Nuclear Physics*, La Jolla, California (2003); <http://www.slac.stanford.edu/econf/C0303241/proc/papers/MOMT004.PDF>.
- [11] S. Agostinelli *et al.*, Geant4 - A simulation toolkit, CERN-IT-20020003, KEK Preprint 2002-85, SLAC-PUB-9350, submitted to Nucl. Instrum. and Methods A.  
<http://wwwinfo.cern.ch/asd/geant4/geant4.html>.
- [12] I. Hřivnáčová *et al.*, in *Proc. of Computing in High Energy and Nuclear Physics*, La Jolla, California (2003); <http://www.slac.stanford.edu/econf/C0303241/proc/papers/THJT006.PDF>.
- [13] R. Brun, A. Gheata and M. Gheata, The ROOT geometry package, NIM **A502** (2003) 676–680.
- [14] <http://www.star.bnl.gov>.
- [15] A. Shoshani, A. Sim, and J. Gu, Storage resource managers: Middleware components for grid storage, in *Proceedings of the Nineteenth IEEE Symposium on Mass Storage Systems and Technologies*, IEEE, New York (2002).
- [16] K. Wu, W.-M. Zhang, A. Sim, J. Gu and A. Shoshani, Grid collector: An event catalog with automated file management in *Proceedings of IEEE-NSS*, IEEE, Portland, (2003).
- [17] LHC Computing Review, CERN/LHCC/2001-004.
- [18] See for instance the results of the LCG Baseline Services study group  
<http://lcg.web.cern.ch/LCG/peb/BS/>.
- [19] See for instance:  
K. Beck, C. Andres, *Extreme programming explained: embrace change*, (Addison-Wesley Professional; 2nd edition (November 16, 2004)), ISBN-0-3212-7865-8.
- [20] <http://alisoft.cern.ch/offline/aliroot-pro/nightbuilds.html>.
- [21] <http://www.cvshome.org>.
- [22] <http://alisoft.cern.ch/cgi-bin/cvsweb>.
- [23] [http://alisoft.cern.ch/offline/aliroot-new/roothtml/USER\\_Index.html](http://alisoft.cern.ch/offline/aliroot-new/roothtml/USER_Index.html).
- [24] W. J. Brown *et al.*, *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis* (Wiley; 1st edition (March 20, 1998)), ISBN-0-4711-9713-0.
- [25] <http://v.mahon.free.fr/pro/freeware/memcheck>.
- [26] <http://valgrind.org>.
- [27] <http://www.intel.com/software/products/vtune>.

- [28] IRST – Trento, Italy is a research centre conducting research in computer science, micro-systems and surface physics. The code checker is now a production tool used by ALICE, ATLAS, IT-Control (PVSS), Root, and others.  
<http://www.itc.it/irst/Renderer.aspx?targetID=111>.
- [29] P. Tonella and Alessandra Potrich, Reverse engineering of the interaction diagrams from C++ code, in *Proceedings of ICSM 2003, International Conference on Software Maintenance*, pp. 159-168, Amsterdam, The Netherlands, September 2003;  
P. Tonella and A. Potrich, Static and dynamic C++ code analysis for the recovery of the object Ddagram, in *Proceedings of ICSM 2002, International Conference on Software Maintenance*, pp. 54-63, Montreal, Canada, October 2002;  
P. Tonella and A. Potrich, Cjj: a subset of C++ compliant with Java, *Science of Computer Programming*, vol. 42/2-3, pp. 229-271, January 2002;  
P. Tonella and Alessandra Potrich, Reverse engineering of the UML class diagram from C++ code in presence of weakly typed containers, in *Proceedings of ICSM 2001, International Conference on Software Maintenance*, pp. 376-385, Florence, Italy, November 7-9, 2001;  
A. Potrich and P. Tonella, C++ code analysis: an open architecture for the verification of coding rules, in *Proceedings of CHEP'2000, Int. Conf. on Computing in High Energy and Nuclear Physics*, pp. 758-761, Padova, Italy, February 7-11, 2000.
- [30] P. Tonella, A. Potrich, *Reverse Engineering of Object Oriented Code*, (Springer; 1st edition (December 17, 2004)), ISBN: 0-3874-0295-0.
- [31] <http://alisoft.cern.ch/offline/codingconv.html>.
- [32] <http://alisoft.cern.ch/offline/aliroot-new/log/violatedRules.html>.
- [33] <http://www.rational.com/uml>.
- [34] <http://alisoft.cern.ch/offline/reveng>.