The ALICE Off-line strategy a successful migration to OO

F. Carminati, R. Brun

CERN Geneva, Switzerland

F. Rademakers

CERN Geneva, Switzerland and GSI, Darmstadt, Germany

For the ALICE Off-line Project – ALICE Collaboration

Abstract

The ALICE experiment has chosen to start developing its software directly in OO, using the services of the ROOT system, which is ALICE's candidate for the common LHC framework. This has lead to the definition of a complete environment (AliRoot) where the software developed by the different experimental groups is being integrated.

Different test-benches for I/O and Simulation have been set up based on real production code. This allows early assessment of technology, both software and hardware, in a realistic production environment. Different codes such as GEANT 4, GEANT 3 and FLUKA, or the reconstruction algorithms by the physicists developing the detectors, can be easily integrated in the framework, which has shown to be both evolutive and modular.

The ALICE Collaboration has adopted this setup and we are now successfully migrating the users into it. This talk describes the AliRoot environment and its future evolution.

Key words: ALICE, CHEP, OO, migration, ROOT

1 Introduction

The development of the ALICE Off-line Framework started at the beginning of 1998. At that time there was no unified framework and the software was composed by a number of independent FORTRAN codes used to prepare the Technical Proposal (TP).

The development of a unified framework had already started a few years earlier for the CMS[1] and ATLAS[2] collaborations. There, the FORTRAN code used for the TP was developed into a full blown FORTRAN environment used also for the sub-detectors' Technical Design Reports (TDRs), while the development of a new environment was started based on the R&D work done in the Information Technology Division[3][4][5], with a longer time perspective. ALICE is a comparatively small collaboration, and it could, therefore, not afford two lines of development. In ALICE code cannot be rewritten, it must be reused.

In ALICE Off-line and *physics performance* groups are merged into a single team. The producers of Off-line code in ALICE are working on the development of the long-term framework but at the same time they have to provide working code to the physics community to prepare the TDRs. This is made particularly difficult as the *official* products for LHC, developed by the R&D collaborations approved by LCB[6] in collaboration with IT were not (and are still not) production ready. On the other hand, the physics user community was not very conversant with C++ and Object Oriented (OO) programming in general. Few users were indeed very advanced, while another small minority was definitely opposed to change, but the large majority of users seemed to be very confused on which direction to take.

This situation posed a substantial challenge to the ALICE Off-line project, not different in essence to the one posed to the other LHC experiments, but made more difficult to face due to the peculiarities of the ALICE experiment. The ALICE Off-line Project had to provide a tool to the user community to design the detectors, whilst developing a framework that could be reused and evolved, and, at the same time, train and involve the user community in this process.

2 ALICE's strategy

The first problem was the development of a coherent simulation framework. GEANT 3.21[7] is a well tested and solid program, but linked to the old CERN Library environment. Its development ended in 1993 and it has been on minimal maintenance since. It has numerous well known limitations in hadronic and muon physics. GEANT 4 is a brand new product, which is still evolving. Its first production release happened in January 1999 and its hadronic part is comparable to GEANT 3. FLUKA[8][9] features excellent physics, and is in fact used for critical studies such as radiation backgrounds and radioprotection, but it is essentially a stand-alone code and rather difficult to use for full-detector simulation, principally due to the limitation in its user interface and geometry.

Faced with this situation we have taken a pragmatic approach based on two decisions:

- There should be only one line of development, forward evolutive, which means based on OO design and implemented in C++
- Only existing working products should be used, even if the ongoing R&D activities should be monitored closely

This resulted in the adoption of ROOT[10] as the base for our framework, and the use of GEANT 3.21 as the detector simulation program. Radiation studies and specialised detector simulation are done with FLUKA outside the framework, for the moment.

As we intended to provide a forward evolutive framework to be used by all ALICE physicists, we have *wrapped* GEANT 3.21 to make it look like a C++ program (class TGeant3). The graphics sub-system from GEANT 3 based on HIGZ has been interfaced to call the ROOT graphics and the interactive interface based on KUIP has been replaced by the more powerful features of the ROOT C++ interpreter. A special version of the CERN Library Routines needed by GEANT has been developed, where in particular the I/O routines of ZEBRA have been removed. All calls to GEANT 3.21 go via an *Virtual MonteCarlo* abstract interface. The GEANT user code has been completely written in C++ and all I/O is performed via ROOT.

This is the only official framework offered to physicists. In order to run simulation or analysis jobs they have to use this OO environment. However, we have decided to allow and help users to interface existing FORTRAN codes with the framework. As of today, ALICE software is almost completely written in C++.

Central distribution and documentation has been organised[11], as well as training courses on ALICE's specific environment. The complete code is available on the web, automatically hyperised, thanks to the ROOT documentation tool.

3 The software development process

The key to the success of this policy is the proper organisation of the software development process. The necessity to provide, in the short term, working tools is considered contradictory with the advantages of proper design. As a matter of fact, this is a surprising opinion held by many experts in the field. Of all analysis and design techniques, OO seems to be the most suitable to fast prototyping techniques, as it allows a very large degree of freedom thanks

to the combined use of inheritance, virtual interfaces, polimorphism and data hiding.

The need to have at any point in time a working tool for production, combined with the constant pressure to evolve and meet the need of the users, has naturally lead us to adopt a fast prototyping – fast feedback software development model, based on existing *off-the-shelf* components.

During the course of the past two years we have progressively adopted a model which is composed of macro-cycles and micro-cycles. During normal development the *new* version of the code is constantly updated by the developers. This is physically achieved via a CVS server accessible remotely by all ALICE members in read only mode and developers in read/write mode. These are micro-cycles possibly discussed and reviewed at the weekly Software Coordination Meetings.

When the development of the code requires major modifications to the structure, a Software Meeting is organised (at CERN or at an external location) and major design choices are taken and implemented. These are macro-cycles that involve step-wise evolutions of the framework. This development model seems well adapted to the needs of an HEP experiment.

During a fast development phase it is necessary to make sure that the code does not *diverge* from the original spirit. A set of coding conventions can be of great help to increase the readability of the code and hence to reduce the time to understand and analyse a new piece of code. After the closure of the Spider[24] IT project, ALICE has started a collaboration with an Italian research institute[26][27] to develop a code checking tool (RuleChecker[28] [29]). This has been written from scratch in Java, based on open source components in three months. The first version of the tool is already in production. A two-year development plan is foreseen, including the development of a reverse engineering tool that will reuse the components of the RuleChecker and will be customisable to understand ROOT containers.

An important part of the model is communication. Mailing lists are extensively used to communicate problems and solutions and detailed information is put on the web, so that the novice user is able to run some simple simulation and analysis examples himself.

Training sessions are regularly organised on framework during major ALICE meetings, open to all ALICE members. These sessions are targeted to the ALICE Off-line environment and have the objective to enable the participants to run simple jobs and modify them to start implementing their own code. After following these courses, most of our users look for, and attend, specialised training in OO analysis and design and in C++ to improve their skills, learn how to use the framework more effectively and participate in its development.

Their reaction to this further training is usually positive and the user community is expanding rapidly within ALICE. This is in striking contrast with the frustration usually reported by physicists who follow the OO courses first, and then are confused on how to translate the notions acquired into practice.

4 The choice of ROOT

The above scenario can only be implemented if there is a robust support for some fundamental functions such as:

- object store and retrieval
- simple 2D and 3D graphics
- interactive data analysis, histograms, *ntuples*-like objects and basic operations on them such as fitting, rebinning and so on
- support for a scripting language

In the absence of this, the only solution is to use PAW[12] for data visualisation, which is intimately linked to the CERN Program Library FORTRAN environment. Without an OO system that integrates all this functionality it is unrealistic to hope to move the user community to OO design and programming. In particular, without the ability to store objects and a seamless interaction between the objects stored and the visualisation tools, the effectiveness of any new system will not exceed the one of PAW, hindering the transition for the normal user.

ROOT indeed provides the needed functionality, and this is the main reason for our choice of using ROOT as the base for the ALICE Off-line framework. However, there are at least two more reasons that make of ROOT the best candidate for our framework.

The development of ROOT is based on rapid prototyping and frequent user feed-back, in a way that much resembles the ALICE software development process. This makes the interaction between the Off-line development and the framework development fruitful and seamless.

The second reason that makes ROOT a formidable help to the development of an OO framework by physicists who are not necessarily knowledgeable in OO analysis and design, is the choice of C++ as a scripting language. The typical development process starts with a simple C++ script that loops over the data. It is quite simple and intuitive for any physicist with some experience in programming to extend the script and make a simple analysis. When this finally becomes a mature program, it can be directly inserted into the existing class structure and be available from a shared library, and the cycle can restart.

The capability of ROOT to be extended with user classes, with an automatic extension of the run time type identification (RTTI), browsing, user interface (UI), I/O and documentation features is also another winning point, which has allowed ALICE to develop with few man-months of effort a complete OO simulation framework.

There is a long-standing debate in the physics community on the modularity of ROOT. The underlying assumption is that modularity is necessary at all levels. We believe that this is a misconception. Modularity means the ability of replacing one part of the code with another one offering a better or different functionality without disrupting the operation of the rest of the code.

But modularity implies granularity, every module in the system is considered as sort of monolithic black-box. In ALICE we have concentrated on the modularity of the user part of the framework, while we are profiting form the tight integration of ROOT containers, I/O and data visualisation. We do not believe that the benefits derived from an increase in the modularity of these components would compensate the increase in complexity coming from the definition of abstract interfaces exposed to the user.

Note however, that this does not mean that ROOT itself should not be or in fact is not modular. The interfacing of ROOT to the RFIO system for instance, developed for the ALICE data challenge, is an example of how ROOT can seamlessly integrate different I/O subsystems without modifying the user application.

Having chosen to migrate immediately and completely to C++, we have been much more worried by the modularity of our simulation system, where we wanted to insulate ourselves from any specific MonteCarlo framework and, above all, we decided not to depend on the time schedule for GEANT 4, and this seems up to now a very wise choice.

5 The ALICE framework

As was said above, the first development of the framework was intended to provide an environment for simulation that could respond to the following requirements

- be a genuine OO framework, including I/O
- exploit GEANT 3.21 initially
- be capable of migrating to GEANT 4 with maximum code reuse

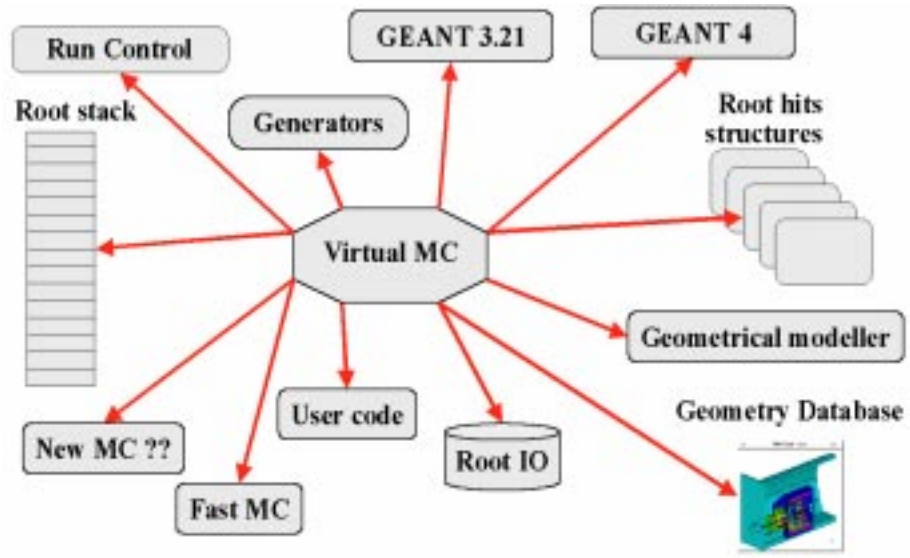

Fig. 1. Structure of the Virtual MonteCarlo

In order to do this we introduced a virtual MonteCarlo interface and wrapped GEANT 3.21 as a derived class from this. All user code is in C++, including the hit generation and recording routines, and all direct dependence on GEANT 3.21 is hidden via the abstract interface as shown in Figure 1. With this simulation code we have performed the majority of the studies for the TDRs and are now preparing to produce the Physics Performance Report.

The main advantage of this system is that while the interactive capabilities of GEANT 3.21 have been preserved, the event loop is now under the control of ROOT and the scripting language is C++. The dependency on the CERN Program Library has been minimised and only GEANT and a small part of ZEBRA (MZ) are required. The KUMAC steering script and the FFREAD data cards of GEANT 3.21 have been replaced by a C++ script, much more flexible and powerful (Figure 3). The replacement of the FORTRAN switch-yards with the virtual function calls (see Figure 2) and of the FORTRAN static libraries with the ROOT-like shared libraries has introduced a large gain, as is reported in Table 1.

Table 1
Comparison FORTRAN and C++ simulation framework

Item	**FORTRAN code**	**C++ code**
Time to link	$<$ 40 sec	$<$ 1 sec
Time per event	710 min	435 min
Size of executable(disk)	11MB	450kB
Size of executable(memory)	24MB	23MB

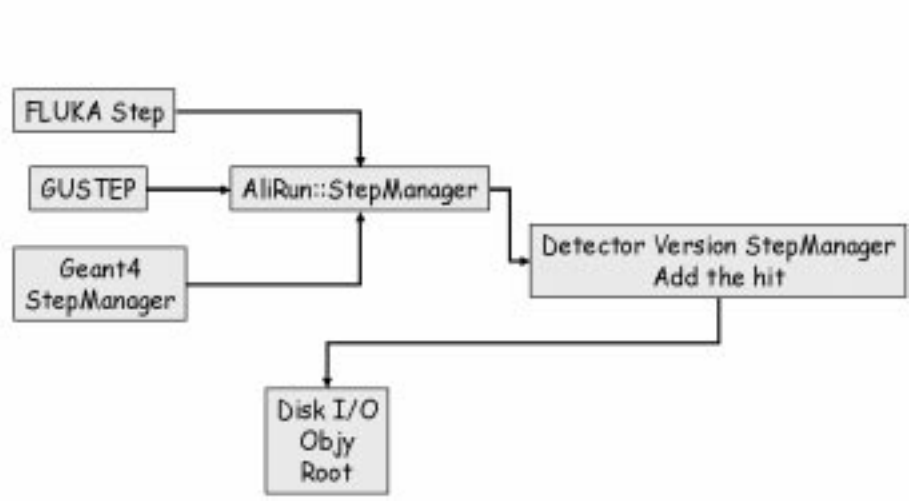

Fig. 2. Stepping routine control in the Virtual MonteCarlo

We have made a large investment to interface GEANT 4 to our virtual MonteCarlo[23], employing 2 people for 2 years. The geometrical interface is almost ready while the interface to the hit generation routines is still under development. This approach turned out to be very effective, as we are directly testing the capabilities of GEANT 4 to provide the functionality needed by a mature production environment.

As far as FLUKA is concerned, in principle it could be integrated in the same environment, even if this will depend on the available manpower. CERN management has announced its intention to provide official support for FLUKA and this may ease its integration in the ALICE framework.

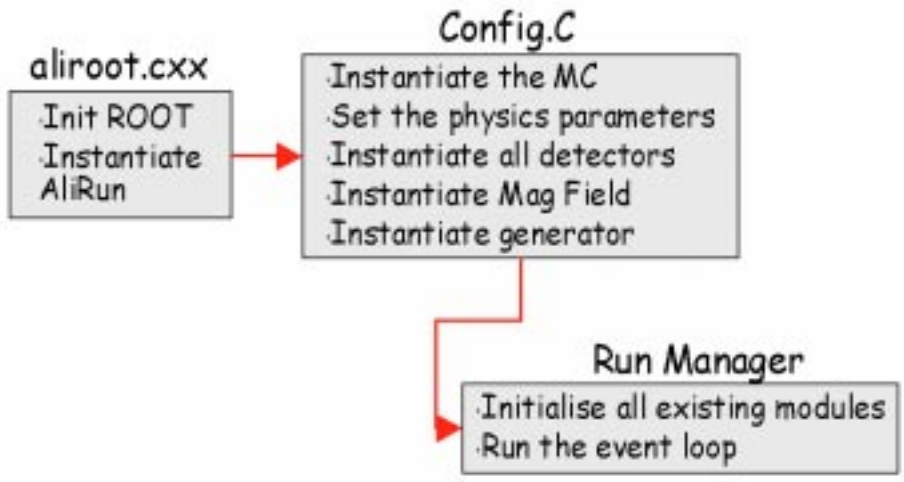

Fig. 3. Control flow of the Virtual MonteCarlo

Having different MonteCarlos in the same framework is the only way to really compare them both for performance and for physics results, eliminating the problem of conversion of the different outputs. At the moment, the ALICE

framework is the only one allowing this kind of comparison.

Our successful experience with the ROOT framework for the simulation naturally lead us to develop reconstruction in the same frame. The resulting infrastructure can be seen in Figure 4.

It can be seen that we can have our output in Objectivity[25], even if, at the moment, we do not plan to use this possibility. We have performed extensive tests of I/O of ROOT data structures with Objectivity, and it is completely feasible, even if the ROOT performance is better in all the cases analysed. The ROOT persistency mechanism is able to directly use our classes and an interactive user can immediately visualise any class attribute via the mechanism of ROOT Trees. An implementation based on Objectivity, in ALICE, would imply the creation of a transient and a persistent object model, making our life far more complicated.

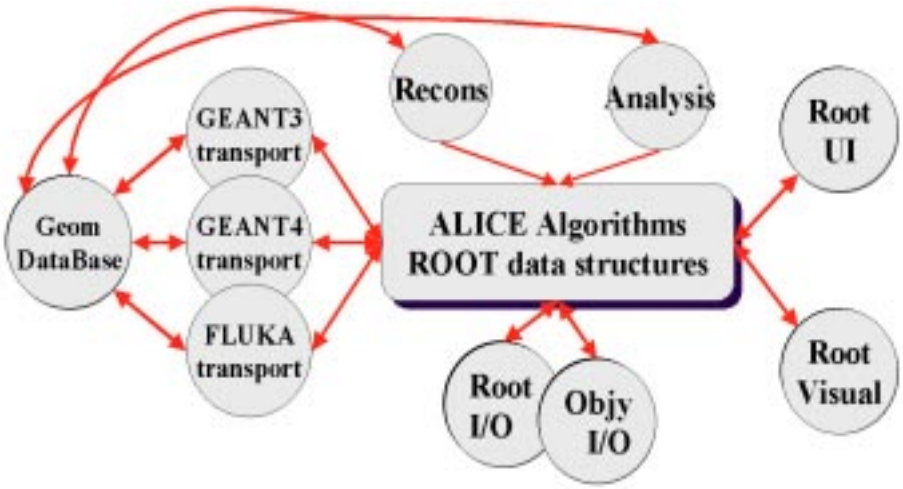

Fig. 4. Structure of AliRoot

In our framework (AliRoot) every sub-detector is a *module* and it is independent from the other modules. Modules contain their specific data structures, i.e. *hits* (precise signals coming from the MonteCarlo), *digits* (simulated or real signals), reconstructed space points, simulated clusters and all intermediate structures. The module also contains the procedures to transform its structures, to build its own geometry and draw itself. For every module there are different versions that are subclasses of the main class. When simulated data is written on disk, the module class is also. When the simulated data is read back, the *right* module class is read back, and the appropriate methods for data handling (reconstruction, visualisation, analysis) are loaded. This greatly reduces the possibility of processing errors, and it is combined with the ROOT schema evolution to accommodate changes in the data over time.

Dependencies between modules have been eliminated and all communications go via interfaces (more or less purely virtual) at the level of the steering mod-

ule. This allows a great flexibility, as the detector can be configured with a simple C++ script. Generators are also loaded via a virtual generator interface and any of the available generators can be loaded at run time. The flexibility of the system is such that even the transport MonteCarlo (GEANT 3 or GEANT 4) is decided at run time via the configuration script.

Data which is not module-specific (*event* data) is handled by global data managers (sometimes called *blackboards*). The general scheme of the AliRoot architecture can be seen in Figure 5.

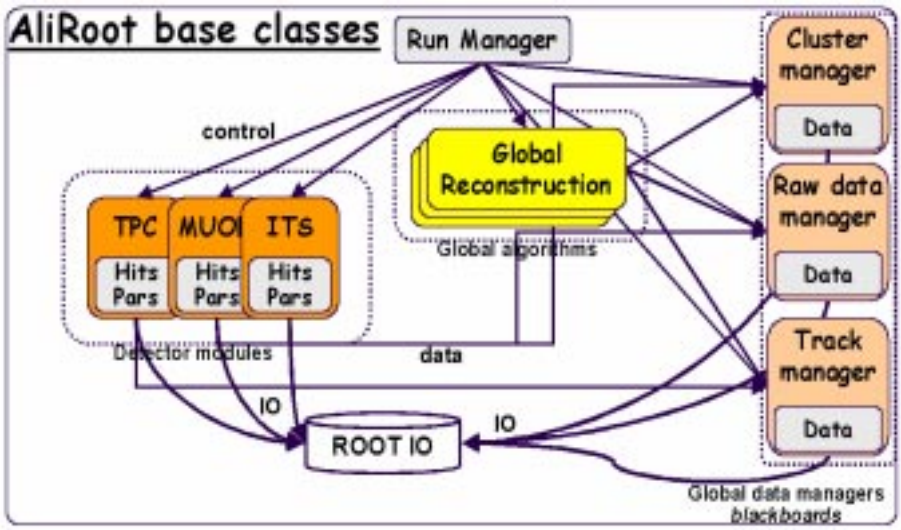

Fig. 5. Architecture of AliRoot

This framework is currently used in production by the ALICE collaboration. It consists of 153kLOC of C++, 39% of which is generated automatically by ROOT. The residual amount of FORTRAN is less than 5% and it is in the process of being translated into C++.

6 The ALICE Mass Storage Project

The storage of data for ALICE is particularly challenging. During ion runs raw data will be taken at a rate exceeding 1GB/s. The combined amount of raw data from proton and ion runs will approach 2PB per year, to which should be added all the derived data and the data coming from simulation. The ALICE Off-line and DAQ projects are looking very closely at the technological alternatives to cope with this situation, both in the area of OO databases and of hierarchical mass storage systems (HMSS).

The RD45 collaboration has recently delivered its conclusions to the LHC Computing Board [6] on OO databases for HEP. The initial confidence that

a specific commercial product could solve all the problems of LHC computing seems now substantially misplaced. The penetration of Objectivity in HEP is still very limited (CMS test beams, BaBar, COMPASS in the near future). The first use of Objectivity in a real production environment has been riddled with serious problems[13]. The development of a home-grown persistent object manager (POM) is considered as a possible alternative to reduce the risks connected with commercial products, but at a very high cost (50 man/years)[14].

The ALICE Collaboration is seriously considering the use of ROOT as an object store, augmented by a simple relational database for the event database. To better assess the situation we have made a series of comparative tests between ROOT and Objectivity [15]. This comparison aimed at evaluating optimisation effort, ease of use and performance for ROOT and Objectivity with a real object model and (simulated) data. Also, we wanted to test the flexibility of our framework.

These tests taught us that cooperation between the two systems is possible. A naive use of ROOT is simple and robust, with performances close to optimal, while a naive use of Objectivity is impossible: some of the features are hardly understood by the experts, and their effect on the behaviour of the system is difficult to evaluate and control. Our test has shown real time and size performance of ROOT as superior to Objectivity.

The other element that we are testing is the HMSS component. For this we have a close and successful collaboration with CERN IT division in the framework of the ALICE Mass Storage Project[16]. The objectives of this process are to assess our computing model in realistic conditions, using high-end off-the-shelf technology. This project is also used to develop integration between DAQ and Off-line, develop an event model and provide a framework to assess new technologies.

The first test conducted in this project was the data acquisition for the NA57 experiment using the HPSS[18] HMSS from IBM. After its successful completion, ALICE decided to conduct a campaign of tests involving commodity hardware and the ALICE Off-line and On-line software - progressively leading to the final system, both in functionality and performance. The first Data Challange was run in April 1998 [19].

After seven days of running with the setup shown in Figure 6 we obtained the results shown in Table 2.

This benchmark tested, for the first time, the DATE[17] (the ALICE data acquisition system), GBit Ethernet, HPPS, ROOT and the ALICE Off-line Framework all together, creating one of the largest OO databases in HEP. This stress-test allowed us to detect some weak points of HPSS that had not appeared before. Raw data was *objectified* by ROOT at 14.7MB/s. This means

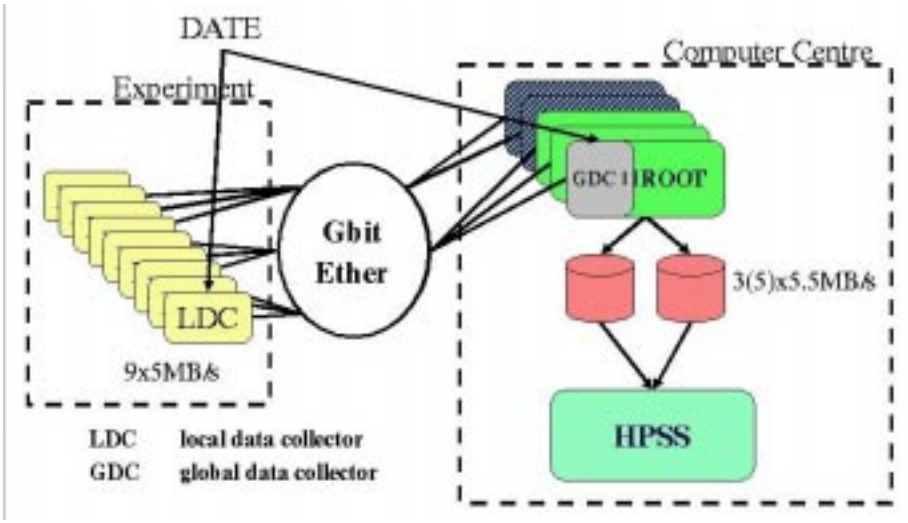

Fig. 6. Setup of the first Data Challange

Table 2
Results of first Data Challange

Number of files	15,436
Number of events	16,229,520
Mbytes in	7,261,382
Mbytes out	6,896,198
Aggregate rate in (MB/s)	14.7
Aggregate rate out (MB/s)	13.9
Total stored to HPSS (TB)	6.9

that with today's technology we could record the ALICE data with 300 high-end PCs, a large but not unrealistic figure.

A second Data Challange is planned in the first quarter of 2000[20] according to the schema of Figure 7. The plan here is to reach 100 MB/s testing both the home grown CASTOR HMSS and HPSS. This will allow us to progressively develop the event model. Instead of random bit patterns as we used in the first Data Challange, simulated digits will be used, and an embryonic level three filter (L3) will be tested.

This test-bench will develop in the future to test different hypotheses of on-line filtering and L3 reconstruction. Different data access patterns will be implemented and evaluated, both locally and remotely. Different HMSS systems will be used in the tests, with the aim of evaluating and comparing them. The next candidate HMSS to be included in the test is Eurostore II[21]. This testbench will also be used as a base for the prototypes of distributed computing to be developed in the context of the Monarc[22] project.

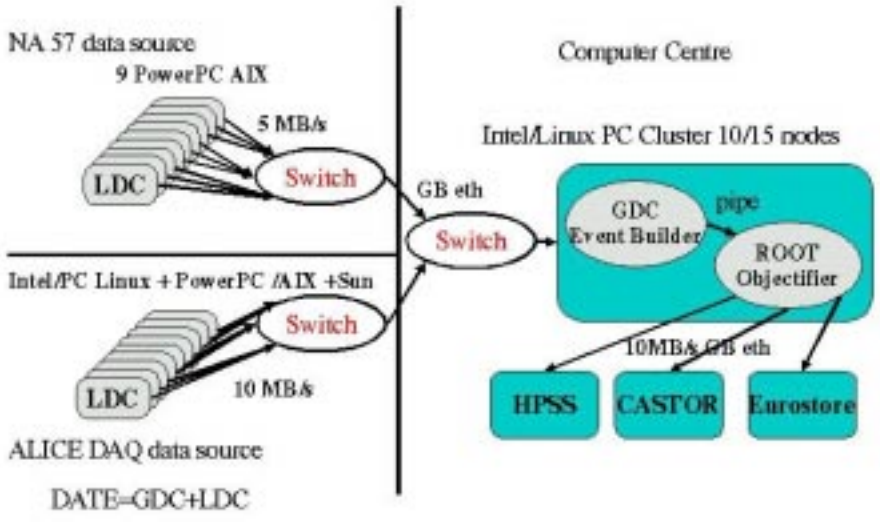

Fig. 7. Setup of the second Data Challange

It is important to underline that all the tests are performed with commodity hardware and production software. This means that they are at any time a snapshot of what could be done in practice at that moment. Improvements and modifications suggested by the tests are directly fed back into the ALICE production Off-line software. This enables us to detect and solve possible problems very early in the development process and it ensures a smooth evolution into the final system giving us us full confidence in its reliability.

7 Lesson learnt

ALICE has been the first LHC experiment to operate a complete transition to OO analysis and design and to C++ for its production framework. Although the construction of the ALICE framework is only at the beginning, this collaboration-wide experience has provided us with a host of information on the transition to C++.

C++ is a very complex language, Stroustrup himself has been quoted with the remark that *nobody understands it all*. It is well known that it is full of redundancies and the code written can be very obscure. On the other hand FORTRAN is simple and well understood by the user community. To migrate to a new environment only makes sense if this is not just better, but also simpler for the user. This requirement may seem to be contradictory with the features of C++.

A no-way-back approach is the only possible strategy. Otherwise users will decide to *migrate tomorrow* and this can go on for decades. But this strategy works only if there is strong central support, proper training and considerable

flexibility to respond to users' needs. Training should target the framework, not the underlying analysis and design philosophy, as physicists should be able to use it as soon as possible. Proper training may follow later. To do this a functional framework has to be present, hence the need to use ROOT as the only functional framework providing the PAW functionality in the OO paradigm and much more.

Initially it was important to accept FORTRAN code for some of the methods, as not everything could be rewritten at once. Far from being a compromise, the re-engineering of this code has been, for the most part, an ideal starting point for people not familiar with C++.

In spite of what has been said and written, users, and in particular remote users, are ready to migrate to OO and C++ if there is a clear infrastructure and a direction for development. Support and consultancy are the key to success here. In ALICE we have invited remote users at CERN for periods of one or two weeks and we have done the work with them, day by day, showing them how to solve their problems in the new environment. These users have been active in AliRoot since, and they have transmitted their knowledge to their colleagues.

Physicists are afraid to be unproductive during a technological transition. If we show them that they can indeed work with the new framework in a more efficient and elegant way, they will be the first to push for change. In ALICE we had several examples of *senior* FORTRAN programmers who have become prime contributors to the new structure. This is even truer for remote users, who will leave their existing environment only when it is clear what the new one is and that there will be support for this.

In some sense modularity is the last thing these users want. Knowing that they can change their histogramming package and I/O back-end at will does not help but rather frighten them. A single integrated environment which is well supported, portable and free is their basic requirement.

Prototyping is an essential part of our strategy and allows us to constantly monitor the evolution of our framework and the feasibility of our requirements. The close collaboration with the On-line group, which has adopted ROOT as the visualisation system for DATE, reinforces this strategy and avoids divergences in the development.

8 A look to the future

The experience of the transition to OO design and C++ has taught us that this is less difficult than anticipated, if the right environment is present. The main issue is perhaps the confidence building process by which users trust a new environment and dare leaving the old one, because they become convinced that there will be support, documentation and help, and that their investment will not be wasted.

The major problems indeed originate from the C++ complexity. These are not bugs, but design features, aimed principally at providing a language that can be extremely efficient and that keeps a very high degree of compatibility with C. We are certainly not the first to discover all this. In fact Java has been developed starting from similar considerations, as a simpler and better C++.

This poses the problem of operating another transition in the future, possibly to Java, in a few years from now. This may be very difficult for several reasons. The code developed will be very large by then, and we cannot think of rewriting it. The users will take it very badly, in the sense that this could break their confidence. A possible solution would be to artificially limit the richness of C++ to a subset that is self-consistent, has all the functionality we need, and we imagine can easily be translated into another idiom. The first part of this work has already been done by the authors of Java, so we are considering the possibility to define a set of coding rules that would make our C++ *Java compliant*. These rules could be implemented as an extension of our existing coding rules. For instance we are already discouraging the use of the C++ templates facility and encouraging the use of polymorphic containers. The Java front-end to ROOT currently being developed in collaboration with FNAL will also help in this possible long term transition process.

If this could be done successfully, we would reduce the complexity of the code without compromising on functionality or performance, and ease the transition to future technologies. This evaluation is already going on as a part of the contract with IRST and we will take a decision on this matter during this year.

9 Conclusion

Thanks to the adoption of ROOT and the decision to make an early migration to an OO framework, we have built a coherent and modular infrastructure. User migration is not a major problem and different components, both FORTRAN and C++, can be seamlessly integrated in our system.

Burning the bridges is necessary at some point to change technology, but this is possible if a robust integrated environment is there **before** asking the users to move, and if enough support and training is provided. AliRoot has already been instrumental in the studies for the TDRs and we are confident it can evolve into a fully blown Off-line framework for the ALICE experiment.

References

[1] CMS Computing Technical Proposal

[2] Atlas Computing Technical Proposal

[3] http://wwwinfo.cern.ch/asd/geant4/geant4.html

[4] http://wwwinfo.cern.ch/asd/rd45/generalInfo.htm

[5] http://wwwinfo.cern.ch/asd/lhc++/index.html

[6] http://www.cern.ch/Committees/LCB/welcome.html

[7] http://wwwinfo.cern.ch/asd/geant/

[8] A. Fassò, A. Ferrari, J. Ranft and P. R. Sala, ‘‘FLUKA: present status and future developments", Proceedings of the IV International Conference on Calorimetry in High Energy Physics, La Biodola (Elba), September 19-25 1993, A. Menzione and A. Scribano eds., World Scientific, p. 493 (1994).

[9] A. Fassò, A. Ferrari, and G. R. Stevenson, ‘‘Forward Shielding for Intermediate Energy Proton Accelerators", Proceedings of the ‘‘Specialists’ Meeting on Shielding Aspects of Accelerators, Targets & Irradiation Facilities", Arlington, April 28-29 1994, published by OECD/NEA (1995), p. 155.

[10] http://root.cern.ch

[11] http://alisoft.cern.ch/offline

[12] http://wwwinfo.cern.ch/asd/paw/index.html

[13] http://lcb99.in2p3.fr/DQuarrie.htm

[14] http://wwwinfo.cern.ch/asd/cernlib/rd45/workshops/july99/espresso/tsld001.htm

[15] ALICE internal note in preparation

[16] http://wwwinfo.cern.ch/pdp/ps/msp/msp.html

[17] http://aldwww.cern.ch/

[18] http://www.sdsc.edu/hpss/hpss1.html

[19] http://root.cern.ch/root/alimdc/alimd-0.htm

[20] http://root.cern.ch/root/alimd100/md100-0.htm

[21] http://www.cern.ch/eurostore/

[22] http://www.cern.ch/MONARC/

[23] http://home.cern.ch/ ivana/AliceG4/G4Main.html

[24] http://www.cern.ch/SPIDER

[25] http://www.objectivity.com

[26] http://www.itc.it/

[27] http://www.itc.it/enITCirst/index.htm

[28] http://zeus.itc.it:4444/

[29] Alessandra Potrich and Paolo Tonella, ‘‘C++ Code Analysis: an Open Architecture for the Verification of Coding Rules’’, to appear in the proceedings of CHEP’2000, International Conference on Computing in High Energy and Nuclear Physics, February 7-11, 2000, Padova (Italy).