Massive-Scale Data Management using Standards-Based Solutions

Jamie Shiers
CERN
Geneva
Switzerland

Abstract

In common with many large institutes, CERN has traditionally developed and maintained its own data management solutions. Recently, a significant change of direction has taken place and we have now adopted commercial tools, together with a small amount of site-specific code, for this task. The solutions chosen were originally studied as part of research and development projects oriented towards the Large Hadron Collider (LHC), which is currently under construction at CERN. They have since been adopted not only by the LHC collaborations, which are due to take production data starting in 2005, but also by numerous current experiments, both at CERN and at other High Energy Physics laboratories. Previous experiments, that used data management tools developed in-house, are also studying a possible move to the new environment.

To meet the needs of today's experiments, data rates of up to 35MB/second and data volumes of many hundred TB per experiment must be supported. Data distribution to multiple sites must be provided together with concurrent access for up to some 100 users. Meeting these requirements provides a convenient stepping stone towards those of the LHC, where data rates of 100-1500MB/second must be supported, together with data volumes of up to 20PB per experiment.

We describe the current status of the production systems, database administration tools that have been developed, typically using the Java binding to the database, data distribution techniques and the performance characteristics of the system. Mechanisms to permit users to find and navigate through their data are discussed, including the issues of naming and meta-data and associated browsers.

Coupled to the data management solution is a complete data analysis environment. This too is largely based on commercial, standards-conforming components with application-specific extensions as required. We discuss our experience in using such solutions including issues related to the integration of several commercial packages. Costs and risk factors are described, as well as issues related to responsiveness of the vendors to enhancement requests and support for additional platforms.

Finally, we discuss the extent to which we have succeeded in our goal of using commodity solutions and the advantages of a database, as opposed to a file-based, approach for the management of vast volumes of data.

Introduction

Traditionally, the software solutions employed at CERN have largely been developed in-house, or within the experimental collaborations. Like many scientific institutes, the programming language of choice has long been Fortran, although C is extensively used for online and systems applications. As we began the planning for the software for the experiments at the Large Hadron Collider (LHC), due to enter operation in 2005, there appeared to be a natural evolution of this software from Fortran 77 to Fortran 90. Indeed, a number of steps were taken in that direction, including an evaluation of which components of the CERN Program Library [1] – a repository for the offline software shared between the experiments – would be redundant with the new standard, or could be simplified by exploiting its features. However, this

migration to Fortran 90 was not to take place and a decision was taken to make no further steps in this direction. Instead, it was agreed to establish a number of projects aimed at identifying and providing object-oriented solutions capable of meeting the requirements of the LHC experiments. This was nothing short of a revolution in our community: indeed, the transition is still under way. Although C++ and Java based solutions have been used in production in a number of areas for several years, it is still true that the bulk of the work is performed using the previous generation of Fortran-based solutions. However, Fortran development is essentially frozen and plans are in place to move to the new tools over the next 1-2 years.

In conjunction with the change of programming language and – more importantly – paradigm, came a growing belief in the use of standards-based, commercial tools. This was not a concept that met ready acceptance within the community – many people were sceptical of the quality and functionality of commercial software and there were considerable concerns over licensing issues. However, it has now been demonstrated that, not only is it possible to produce production systems based on commercial solutions that are capable of meeting our requirements, but also that the licensing issues can be solved, both for CERN and for collaborating institutes.

This paper describes the work of two of the projects that were set up as part of the overall plan to move to object-oriented solutions. These projects were established to provide a C++-based "equivalent" of the CERN Program Library, and to solve the data management problem respectively. Both projects – LHC++ [2] and RD45 [3], were given strict guidelines to investigate the use of commercial tools and have reached the stage of offering production services. A third project, GEANT-4 [4], aimed at providing a toolkit for the simulation of the complex detectors used in HEP, is based on a common strategy – i.e. it makes use of a consistent set of underlying class libraries and is based upon the same standards. In this case, however, it was clear that a commercial solution was not appropriate and a world-wide collaboration was established to design and implement a package capable of meeting the requirements of the new experiments. It is believed that these projects not only demonstrate the successful use of object-oriented methods on a large scale, but also the use of off-the-shelf solutions in areas that they are applicable.

The extent to which it is possible to build large-scale data management, analysis and visualisation systems upon off-the-shelf commercial components, the associated benefits, drawbacks and risks, is discussed below.

Overall Requirements

The overall requirements that we had to satisfy can be classified in a number of ways, including in terms of scalability and functionality, which we discuss in this section.

The term "scalability" is often misused as a synonym for "the level at which a system must work up to" – e.g. the maximum number of files, total amount of data and so forth. In some senses this is natural, as it is at the upper end of the range of scalability that most problems occur. More correctly, it should be used to refer to the range over which a particular solution is applicable. In our environment, we typically have to deal with a very wide range of scalability. For example, solutions to problem of object persistency had to be found that were applicable both to small objects, e.g. histograms, tracks, etc., but also for very large (composite) objects. In other words, a system was required that scaled from perhaps 100 bytes per object to objects of 100MB in size and meanwhile satisfy storage needs from under a GB to 100PB. Similarly, the solutions needed to work not only at a site such as CERN, with many thousands of client machines, but also at collaborating institutes with just a few users working on a small fraction of the data.

Another requirement that had to be addressed was that of adaptability – the overall time scales for the LHC are so long that no single solution can be assumed to survive. Thus, a strategy had to be established whereby individual components could be replaced by others whilst preserving the functionality of the whole. This is an issue that should not be dealt with too lightly – we simply do not know what changes the future will bring and hence "planning" for such changes is somewhat difficult. Perhaps the best handle

that we have on the future is the past. This will not tell us which changes we will see, but gives some indication of the magnitude of such changes. For example, just ten years ago, when the Web had still to be invented, mainframes dominated CERN computing. Only a few years later, the migration to Unix and distributed computing was complete. Then a migration to NT began, somewhat haltingly, and now seems to have changed direction towards Linux. However, the end of LHC data taking and analysis might be 25 or more years in the future. This is an extremely long time in the computing industry and not so short in terms of a career!

Software Requirements

Aside from the need for object persistency, discussed in more detail below, the overall software requirements included the need for foundation libraries, mathematical libraries, graphics, visualisation toolkits and more HEP specific areas, such as detector and event simulation packages, routines for relativistic transformations and so forth. Clearly, in many of these areas there is a wealth of commercial software available. On the other hand, the market in the more specialised areas is so small that in-house development is required.
Given that the overall environment would then be composed of multiple packages from different sources, issues related to compatibility and interoperability had to be addressed. Thus, our strategy was to identify standards and commercial components that could meet our more general requirements, whilst in parallel defining those areas where HEP-specific solutions were likely to be needed. In addition to the overall functional requirements, it was considered essential that the software be of the highest possible quality. At the LHC, data will be selected using a series of filters, which will be used to select only data of interest – reducing significantly the volume of data that is recorded. The fact the software will be used in this process has sensitised people to the importance of correct operation – if the software is buggy, data will be irretrievably lost. Of course, it is equally important that the hardware functions correctly, but as it is relatively new that software be used as part of the data selection – or rather rejection – process, the importance of correctness of operation is given much emphasis.

Standards

It is our belief that the use of widely-used, standards-based solutions not only results in a lower maintenance load but also increases the chance of a smooth migration path in case of follow-on products. In principle, at least, widely used software should be well-tested and also more affordable, as the cost of development has been amortised over large numbers of users. We therefore attempted to identify the relevant standards – either de-facto or de-jure. In addition to the standards being developed by the SSSWG, these included:

- The Object Database Management Group (ODMG) [5] standards for Object Databases (ODBMS),
- OpenGL [6], OpenInventor [7] and VRML for graphics and virtual reality,
- Rogue Wave's Tools.h++ [8] class library,
- The C++ standard and the STL, expected to replace Tools.h++ in many areas.

Some of the advantages of using standards are highlighted by the choices of graphics standards above. OpenGL is very widely adopted: support is provided by all of the major vendors and there are many inexpensive graphics cards that offer hardware acceleration. Alternatively, as was typically done in the HEP community in the past, one can develop or adopt proprietary or exotic solutions and suffer from lower performance and lack of support. Similarly, there is expected to be migration path from OpenInventor to a possible follow-on product – Fahrenheit – a joint project between HP, Microsoft and SGI.

Coupled to the above, we chose the NAG mathematical libraries [11], which largely met our requirements in this area, and the IRIS Explorer visualisation system [13]. The former can be considered an industry

standard – it is available at a large fraction of scientific institutes worldwide. OpenGL and OpenInventor both originated at SGI, as did IRIS Explorer, The latter is built upon these two graphics packages and is now maintained by NAG.

Thus, the overall strategy is consistent and offers numerous advantages, not least of which is the fact that the number of suppliers are minimised, making interoperability is easier and allowing solutions to be reused. For example, HEP extensions built on top of OpenInventor for use in "stand-alone" mode can be easily incorporated for the IRIS Explorer system.

HEP Extensions

There were a number of areas where the products chosen did not fully meet our requirements. Rather than simply develop our own "standard", we have preferred to build on the top of standard components. It is important to point out that the extensions that were required have typically been modest – in many cases they represent some 2-3K lines of code and in some cases much less.

In a number of cases, these enhancements will be added to future releases of the products in question. For example, a small number of special functions, such as Bessel functions for complex argument, were not provided by the NAG libraries but were required by certain applications within our community. The relatively short list of functions has been identified, and a future release of the NAG C library will incorporate these functions. Similar enhancements have occurred in other products, such as IRIS Explorer, where developments originally made at CERN have found their way into the standard product.

The success of this strategy clearly has an important impact on the maintainability of the overall system. Even if the various HEP extensions were not to find their way back into the corresponding products, the support load that is offloaded by using standard components is very significant. Naturally, there are drawbacks too: effort is required to ensure that the various products interoperate correctly and are supported on the same combination of operating systems and compilers – a non-trivial task, as vendors struggle to implement the latest features of the C++ standard. Again, minimising the number of suppliers involved has a positive impact in this area.

Data Management Choices

As previously described in this conference series [17], the data management system will be built on a combination of two commercial products, namely Objectivity/DB and HPSS. The interface between these two products is the subject of another paper at this conference [18] and will not be discussed further here. The choice of these products was mandated by the overall requirements in terms of scale and functionality. The total data volume that needed to be supported was in the region of 100PB – over a very long time period – giving rise to the need for a highly scalable mass storage system. In addition, given that the data to be stored were in the form of objects – C++ and/or Java – a mechanism of providing object persistency was required.

The rationale behind the decision to choose an ODBMS, as opposed to potential alternatives such as language extensions, persistent object managers, relational systems with object bindings etc., was described in a paper submitted to the 15th IEEE/6th NASA Goddard Mass Storage conference. Fundamentally, the decision was driven by the requirements of performance and scalability, plus of course functionality – which it is believed that none of the alternatives can adequately address. At a very high level, the functional requirement was to provide access to any object, or the data members of any object that resided in a multi-PB store in an efficient manner.

In addition to the above, the use of an ODBMS together with a language such as C++ or Java provides a very natural way in which persistent objects can be managed. Indeed, this was one of the goals of the

ODMG standard – to provide an environment that appeared to the user as a natural extension of the programming language, and not, as is the case with, for example, embedded SQL, two markedly different environments.

In other words, if your data model is in fact an object model, an ODBMS makes a lot of sense. If, in addition, you have a requirement for distributed databases with, for example, consistent, shared, schema, and scalability beyond some tens of GB, you will be led to a certain product choice. If your data model is not object oriented, or if you have only modest data volumes, then alternative approaches, including other ODBMSs, RDBMSs with or without object oriented bindings, may be appropriate. However, it is perhaps worth considering that applications often grow in terms of their requirements and rarely, if ever, shrink. Thus, although it may appear today that you have no need for a distributed database and/or have only small volumes of data to manage – the latter presumably being not the case for attendees at this conference – longer term needs might suggest a more scalable approach.

Similarly, if your data volumes are less than a few TB, a disk-based solution may well be sufficient, obviating the need for a mass storage system. Again, this is presumably not the case for sites represented. Given the flexible design of the interface between Objectivity/DB and HPSS, alternative mass storage systems may be used. Indeed, the interoperability of Objectivity/DB and the Sony mass storage system was described at the Computing in HEP conference in Chicago 1998 [16].

Status of the Projects

After some three years of research and development, the conclusions of the projects has been that it is indeed possible to build a production quality system out of off-the-shelf commercial solutions. Typically, the extensions that are required are minimal, although in certain cases there is no realistic alternative to in-house development.

Although the initial goal was to address the needs of the LHC experiments, the solutions that have been identified have been adopted by a number of existing experiments. In addition, even though data taking at the LHC will not commence until 2005, there is a considerable amount of work already under way to design, build and test detector elements. Production services based upon Objectivity/DB and HPSS have been setup and production releases of the overall software environment have been made. These have been used in a number of test-beam activities, both for ATLAS and CMS (two of the LHC experiments), and for experiments that will take considerable quantities – up to several hundred TB/year – of data from 1999 on.

Two such experiments are COMPASS and NA45 at CERN – several others exist at other HEP laboratories, such as the BaBar experiment at SLAC, with whom there is close collaboration and joint development on both HPSS and Objectivity/DB issues. Indeed, the Objectivity/DB – HPSS interface has recently been used to store over 1TB of data, the bulk of which is transparently moved to HPSS-managed storage and hence migrated to tape, without requiring changes to database applications.

The two CERN experiments mentioned above will take data at rates of 35 and 15MB/second respectively – tests of such data rates have already been demonstrated. The COMPASS experiment anticipates some 360TB of data per year – data volumes and rates not dissimilar from those expected at the LHC. Thus, in many respects, the problem that we have to solve is not in 2005, but now.

Risk Factors

Although there are clearly risks associated with any solution, it is our belief that the use of widely used commercial components offers better protection – particularly given the extremely long timescales of our projects – than developing everything in-house. Fundamentally, it depends on whether there is a

sustainable market for solutions that can satisfy our needs. It is clear that one cannot predict if a particular product will survive, but if there is a long-term need, then a solution will be provided.

In many areas, we believe that the risks are low – even negligible, compared with other uncertainties, such as long-term funding. For example, one can be confident that high-quality graphics will still be available in 10-20 years time. The implementation will almost certainly differ from what is available today – it is virtually guaranteed that future systems will be significantly more powerful than those that are available today. Similarly, the need for mathematical software will continue to exist. Here, the market is clearly much smaller than for graphics, but it is nevertheless sufficiently large as to support a number of companies and a wide range of solutions.

On the other hand, the risks associated with the data management software are much larger.
The market for high-end mass storage systems continues to be small. Even though the volumes of data that will be stored in the future will increase, there is no convincing evidence that suggests that there will ever be "commodity" mass storage systems.

Similarly, the market for ODBMS products is growing much slower than was predicted. Increasingly, even object-oriented applications are being built on top of relational databases, including so-called object-relational systems. Although the advantages of true object databases are significant – both in terms of performance and scalability – there seem to be relatively few applications that need, or believe that they need, these features. There are a number of projects that require support for very large volumes of data, but most of these, if not using more traditional approaches, such as storing meta-data in a database and the data itself in files, are in the scientific market place. Outside the HEP community, projects with similar requirements exist in astrophysics, geophysics and plasmaphysics.

Certainly, no-one is predicting that object databases will replace relational databases in the foreseeable future. Is the market that exists, which appears to be dominated by "technical" computing, such as telecoms, scientific etc., large enough? Current estimates suggest that the global market for satellite-based communications systems, such as IRIDIUM, will be as large as $40bn by 2010. Given that – at least in the case of IRIDIUM – the software involved is built on an ODBMS, there is reason to believe that this market will survive. However, will solutions that meet the needs of the telcoms industry scale sufficiently to handle the requirements of "big science"? If not, could multiple independent databases be used in a manner that is both manageable and sufficiently functional? Unfortunately, the answers to these questions are not yet clear and so more work in the area of risk analysis, plus the development of a fallback solution, is required.

Migration Strategy

Given the timescales involved, it is inevitable that one or more – if not all – components will change. This is in sharp contrast with the previous situation in HEP. An experiment was able to choose an environment and stay with it throughout its entire lifetime. This will clearly not be possible at the LHC. One of the potential advantages of using standards is that there will often be a migration path to "follow-on" products. For example, it is possible that the HP/Microsoft/SGI "Fahrenheit" project will produce products that will replace OpenGL and OpenInventor. It is expected that the APIs of such products will be, to a large degree, compatible with those of today, allowing a smooth migration. Furthermore, it is possible that the changes could be made transparent to the user via the existing HEP layers that have been developed.

In some cases, however, the "new" and "old" are based on different paradigms. This is the case with Rogue Wave's Tools.h++, that provided a de-facto standard for collection classes, and the STL. The differences between these two libraries are much more significant than is expected between e.g. OpenInventor and Fahrenheit.

However, we have already demonstrated a successful migration of individual components of our strategy – and software developed thereon: in this case from Tools.h++ to the STL. Furthermore, we have also demonstrated that modules developed for one visualisation system can be implemented relatively easily in another, reusing the bulk of the code.

Other changes that may take place include that of the programming language. Although the solutions provided today are based upon C++, there is growing interest in Java, which is already being used in a number of areas, such as for a database administration tool. These languages are likely to coexist for some time, and hence interoperability is of importance. For example, persistent C++ objects that are stored today should be usable from Java – or perhaps an entirely new language – in the future. However, the features of these languages – let alone once that have yet to be invented – are such that full interoperability is unlikely. Nevertheless, within certain constraints, we believe that an acceptable degree of interoperability between at least C++ and Java at the level of the ODBMS will be possible.

The Production Chain

In a somewhat simplified view, data are acquired from the detector (real data), or the response of the detector to simulated events is performed (data generated according to some theoretical model). These data are then converted from raw readout signals to particle tracks and energy clusters and finally analysed. The specific requirements of the various stages are described in more details below.

Data Acquisition

The most significant challenge regarding data acquisition is that of data rate. Experiments today need to be able to store 5-35MB/second and future experiments will acquire data at rates ranging from 100MB/second to 1.5GB/second. To handle these rates, the use of parallelism is required. The data are acquired several kilometers from the computer centre, where there are written into a mass storage system. Thus, multiple parallel network streams are used, if needed to multiple database servers. Demonstrations by existing experiments have already demonstrated that rates of 35MB/second can be sustained. This suggests that rates of 100MB/second, as required by several of the LHC experiments in 2005, will be relatively easy to handle.

In fact, tests using a HP Exemplar system at Caltech have already shown that data rates up to ~150MB/second can be handled with the appropriate resources, although rates in excess of 1GB/second still require further study. Typically, data are written into a file or database that resides close to the detector. At a certain stage, e.g. when the filesize approaches a limit, the file is transferred over the network and written into the mass storage system. A local buffer is maintained such that network outages of several hours can be tolerated.

Reconstruction

The reconstruction process is highly CPU-intensive. In order to keep up with the rate at which data is acquired, this stage is also performed in parallel. Current plans for LHC computing call for 100-500 processors in a reconstruction farm, each working on individual events. Tests of up to 250 processors have already been made, suggesting that this requirement can be met, even today. Each processor needs to support a relatively modest data rate – a few MB/second, although the overall data rate is of course much higher.

Analysis

The analysis stage is much harder to quantify than the previous two stages. Typically, only a subset of the data is involved in any given analysis. That is, only a subset of the total event sample is of interest and only a small amount of the data per event is required. Thus, a system that readily supports access to a subset of the data has the potential of greatly reducing the time required to perform a given analysis. This has already been demonstrated: certain analyses are performed many times faster if the features provided by the ODBMS for random access are properly exploited. However, the performance improvement that is obtained depends very much on how the data are clustered. Unfortunately, many different access patterns need to be supported and, without storing the data multiple times, it is not possible to cluster the data optimally for all cases. However, ODBMSs offer great flexibility in data clustering and, at a certain cost, allow the data to be reclustered later.
Clearly, the goal is to perform an initial clustering that is at least adequate, and preferably optimised, for the most frequent access patterns. Given the cost of reclustering, it is perhaps most appropriate to "recluster" the data when it is reprocessed, e.g. if new or corrected algorithms are discovered, or if better calibrations for the detector are available.

A possible analysis scenario for the ATLAS experiment at the LHC is as follows:

- Some 20 working groups, each consisting of 10-30 people, are involved in the analysis of the data at any one time,
- These working groups correspond to approximately 150 users accessing the data concurrently (tests have already demonstrated some 250 "users" accessing the database),
- Each analysis group looks at an event sample of some 10^9 events, producing sub-samples for further analyses,
- These sub-samples are then analysed 10-15 times per day.

Given that the total size of an individual event for the LHC experiments is approximately 1MB, it is clear that the above scenario is highly impractical if the full event data is read.
For example, if each of the 20 working groups processes an event sample of 10^9 events once per month, this is approximately equivalent to an event sample of 10^9 events being processed once per working day. This would require a data rate of 10GB/second – in addition to the data rate required for data acquisition, reconstruction, reprocessing, reclustering and, last but not least, the 10-15 passes per working group per day that are made over the subsets extracted from these event samples!

Thus, it is clear that the analysis stage will impose the most significant requirements on the overall system.

In order to optimise the analysis phase, various strategies are currently being investigated.
These include the use of a so-called event tag, in which the key characteristics of each event is stored. Some analysis will be possible just using these tags, which are expected to be of the order of 100 bytes in size. Other analyses will perform a first selection using the information in the tag, and then use ODBMS features to transparently navigate from the event tag to the main event data. Again the capabilities of the database will be exploited to ensure that only the needed data is read – a very significant advantage over what is possible with conventional, file-based systems.

Data Analysis Environment

The data analysis environment that is currently being developed is based upon the commercial components mentioned above, including IRIS Explorer and Objectivity/DB, with HEP-specific extensions as required. IRIS Explorer is a modular visualisation toolkit, similar to other well-known tools such as AVS. Of particular interest to us was its use of OpenInventor graphics and NAG mathematical routines. In common with other such systems, an application is built graphically, out of individual components. A wide range of components is provided with the system. These can be further extended with public-domain components and those written in-house. The various modules that make up an application communicate

efficiently via shared memory – or sockets if the modules are run on different machines. The latter option can be usefully exploited, say, to run a I/O intensive module "close" to the data on which in is working, whilst displaying the results locally. Similarly, IRIS Explorer comes with built-in support for a variety of data types, and external data formats, such as netCDF and HDF.

Given our choice of Objectivity/DB and HPSS as the basis for a data management solution, an interesting question was the integration of an ODBMS with IRIS Explorer.

This was accomplished in two ways. Firstly, a set of modules was provided that allowed users to select data from the database. Secondly, the set of IRIS Explorer data types was extended with a persistent object identifier (OID). This then allowed modules to exchange OIDs, whilst the database maintained consistency and integrity.

Data Selection

Typically, a high-level selection is made by choosing a *collection*. This is implemented as a set of object references (OIDs) using the semantics of the Standard Template Library (STL). Thus, whilst many of the STL algorithms are inapplicable to large collections of data, an interface is provided such that the meaning algorithms can be applied. Typically, a collection is entirely decoupled from the physical organisation of the data. That is, the objects referenced from a collection can have an essentially arbitrary clustering. However, in certain circumstances, a coupling to the physical clustering is appropriate.

Collections may overlap with each other and, if implemented using bi-directional associations, be automatically updated should the referenced data change.

Once accessed, a user would typically iterate over the collection, applying additional selections and deriving data as required.

Collections can be named – using a familiar hierarchical naming scheme, i.e. based on that of the Unix filesystem and have associated meta-data. At this stage the meta-data is simply composed of an object consisting of tag + attributes pairs – essentially the same as is used for the event tag described above, but with the possibility to store also character fields. More sophisticated meta-data objects are planned, allowing, for example, the processing history of an event collection to be associated with it.

Again, these features are relatively easy to implement using features of the underlying system – in this case an ODBMS.

Distributed Data Analysis

Up until now, it has been assumed that the data processing will occur at a single site. However, this is unlikely to be the case. Although the data acquisition, and perhaps also reconstruction, is likely to be performed centrally, the analysis and also the generation of simulated events will almost certainly be performed in a distributed fashion. This brings with it many new questions – should we establish a single, world wide distributed database per experiment? Is this even feasible – regardless of network and other such constraints – considering the implications that this would have on the management of the installation and upgrade of compatible versions of the ODBMS itself, plus operating systems and compilers? Alternatively, should we use independent databases and provide tools for data import and export? The answers to these questions are not yet known: in the short term, network bandwidth probably rules out the use of wide-area distributed databases. However, regardless of the implementation, there is still a requirement for users to be able to perform analysis on a data sample when they, or their data, are located at CERN, at their home institute, or even in transit. Clearly, this is an area where further work is required and a new project to address these issues has recently been launched.

Conclusions

A powerful, modern data management and analysis environment has been built out of a small number of widely used, standards-based commercial solutions, coupled with a small amount of application-specific code. It is believed that the use of standard components with minimal site-specific extensions offers the best protection against the many changes that will inevitably occur over the lifetime of the LHC. In addition, the functionality offered by the overall system is much greater than that provided by "legacy" systems, whilst also supporting much higher data volumes and rates.

The system is being used in production today, and is capable of handling data volumes and rates ranging from a few GB and KB/second up to those expected at the LHC at CERN.

Acknowledgments

The work described in this paper owes much to the efforts of many people, including those involved in the LHC++ and RD45 projects, the IT/ASD group at CERN, and the many people collaborating with the above.

References

[1] CERN Program Library, see http://wwwinfo.cern.ch/asd/index.html.
[2] Libraries for HEP Computing (LHC++) - see http://wwwinfo.cern.ch/asd/lhc++/index.html.
[3] RD45 - A Persistent Object Manager for HEP, LCB Status Report, March 1998, CERN/LHCC 98-11. See http://wwwinfo.cern.ch/pl/rd45/reports/sr_98/status_report.html.
[4] GEANT-4 - http://wwwinfo.cern.ch/asd/geant/geant4.html
[5] The Object Data Management Group (ODMG), see http://www.odmg.org/.
[6] OpenGL Reference Manual, ISBN 0-201-63276-4, Addison Wesley, 1992.
[7] OpenInventor Reference Manual, ISBN 0-201-62491-5, Addison Wesley, 1994.
[8] Rogue Wave - see http://www.roguewave.com
[9] ObjectSpace Internet<ToolKit> - see http://www.objectspace.com/.
[10] CLHEP - A Class Library for HEP, see http://wwwinfo.cern.ch/asd/lhc++/clhep/index.html
[11] NAG C and Fortran libraries - see http://www.nag.co.uk
[12] Minuit - A Function Minimization and Error Analysis Package, CERN Program Library D506. See http://wwwinfo.cern.ch/asdoc/minuit_html3/minmain.html.
[13] IRIS Explorer User Guide, ISBN 1-85206-110-3, 1995. See also http://www.nag.co.uk/Welcome_IEC.html..
[14] STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library, David R. Musser, Atul Saini, Addison Wesley, ISBN 0-201-63398-1
[15] 3DMaster Suite - see http://www.tgs.com
[16] Computing in High Energy Physics 98 (CHEP98), see http://www.hep.net/chep98/.
[17] Building a Database for the LHC – the Exabyte Challenge, Jamie Shiers, CERN. In proceedings of the 15th IEEE Symposium on Mass Storage Systems.
[18] Coupling a Mass Storage System With a Database for Scalable High Performance, Andrew Hanuchevsky (SLAC), Marcin Nowak (CERN). In proceedings of the 16th IEEE Symposium on Mass Storage Systems.