LHC Computing Review

LHCb answers to the SPP questions

Document Version:	0.4
Document Date:	22 February 2000
Document Status:	Draft
Document Editor:	Pere Mato

Abstract

This is a working document for collecting the answers to the proposed questions of the Software Project Panel (SPP) for the LHC Computing Review.

Disclaimer

This document is a progress report towards answering the questions addressed by the software panel. We have not had the opportunity of consulting widely with our colleagues and we therefore reserve the right of modifying and developing the answers further.

1 Process, Planning, Training and Milestones

1.1 Which elements of the Computing and Software Organization participate, and interact, to effect the design and development of the software?

The mandate of the LHCB Computing team is to take responsibility for the computing infrastructure, which includes both hardware and software aspects, for all computing intensive activities of the experiment. This includes computing for the on-line and off-line systems, support for office automation and documentation, tools for enhancing effective world-wide collaboration etc. The project organisation takes into account the interface with the people that take responsibility for sub-detector dependent software and hardware developments. By organising all activities under one organisation we aim to minimise unnecessary duplication and make efficient use of our resources e.g. between DAQ and controls, as well as between online and offline (reuse of software).

The process for organising development of LHCb software is shown in Figure 1. We concentrate our efforts on producing high quality components that can be reused wherever possible in different applications (*Build*). Higher quality implies that greater effort must be put into their development and that more rigorous methods are followed to ensure quality is maintained. The extra effort that this implies can be compensated by the ability to reuse well designed modular components wherever possible in order to produce the final data processing applications (*Assemble*). Underlying the process are the methods and tools used to organise and develop the software, to manage the code repository and software releases and to manage the documentation (*Support*). Project management techniques are used to initiate software projects, to track their progress (e.g. through review) and to assign resources. (*Manage*)

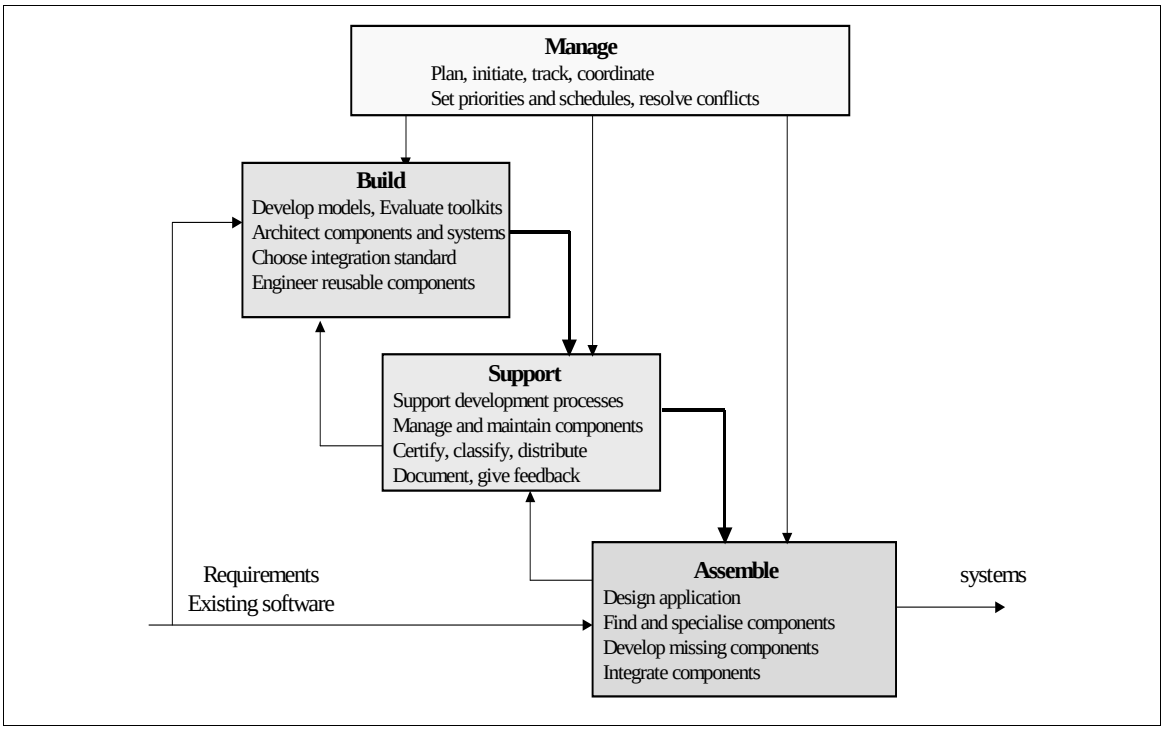

Figure 1 Process for organising LHCb software development activities

A project management structure has been set-up in LHCb in order to encourage these goals to be achieved naturally (Figure 2). Initially attention has been given to defining the architecture of the software and to building a software framework that ensures this architecture is respected (GAUDI). Projects are envisaged for all the various on- and off-line application domains and activities are well advanced in DAQ and in Controls. A migration strategy has been defined for evolving the existing FORTRAN based simulation and reconstruction codes to use the new OO framework and the new reconstruction program (BRUNEL) will shortly replace the old software (SICb) in production. New frameworks for analysis and simulation are planned for the next 12 months.

We are convinced of the importance of the architecture and appointed an architect to lead the project. The architect needs to have a combination of skills:

- software engineer - designer and technologist (OO mentor)
- physicist - knowledge of data processing applications
- manager - form, lead and inspire the design team
- visionary - have picture of what architecture should look like

We started with a small design team 6-8 people, incorporating domain specialists experienced in design and programming. The software librarian was a member of the core team from the beginning. At the beginning there was a lot of brainstorming and it was important to control activities through visibility and self discipline. The core team met regularly, in the beginning every day and then twice per week as ideas became consolidated and people started to use a common vocabulary. Now the team meets once per week.

One member of the design team was devoted to collecting use-cases and these were used to drive the development and to validate the design. The basic design criteria were soon established for the overall architecture, defining the architectural style, flow of control, and the model for specifying interfaces.

Our software process corresponds well to the USDP approach i.e. it is architecture-centric (namely GAUDI), iterative and incremental. Since starting in fall of 1998 we have made public software releases three times per year. Each release is timed to coincide with an LHCb software week, during which new features are explained to users through presentations and hands-on tutorials. During these weeks priorities for features to be included in the next release are discussed together with physicists using the code. Intensive use of the GAUDI framework started after the third release in Nov 1999 once most of the required functionality had been included.

From the beginning we have had a software librarian as a member of the GAUDI team responsible for managing the code repository and making software releases. Also effort has been put into designing the LHCb web and on developing tools for managing collaboration information (e.g. BWHO, document browser). We have devised C++ coding conventions and have organised training in programming in C++ and into techniques for OO analysis and design. However there has been no formal approach to adopting design methods and tools, nor procedures for quality control etc. This has been due to lack of manpower and not because we perceive them to be unimportant.

The project leaders responsible for the various computing subsystems (*Steering Group*) meet together on a monthly basis thus ensuring an overall co-ordination of both on-line and off-line computing. Reviews have been held for GAUDI and recently a series of reviews have been started for the newly developed OO pattern recognition code for the various subdetectors

.

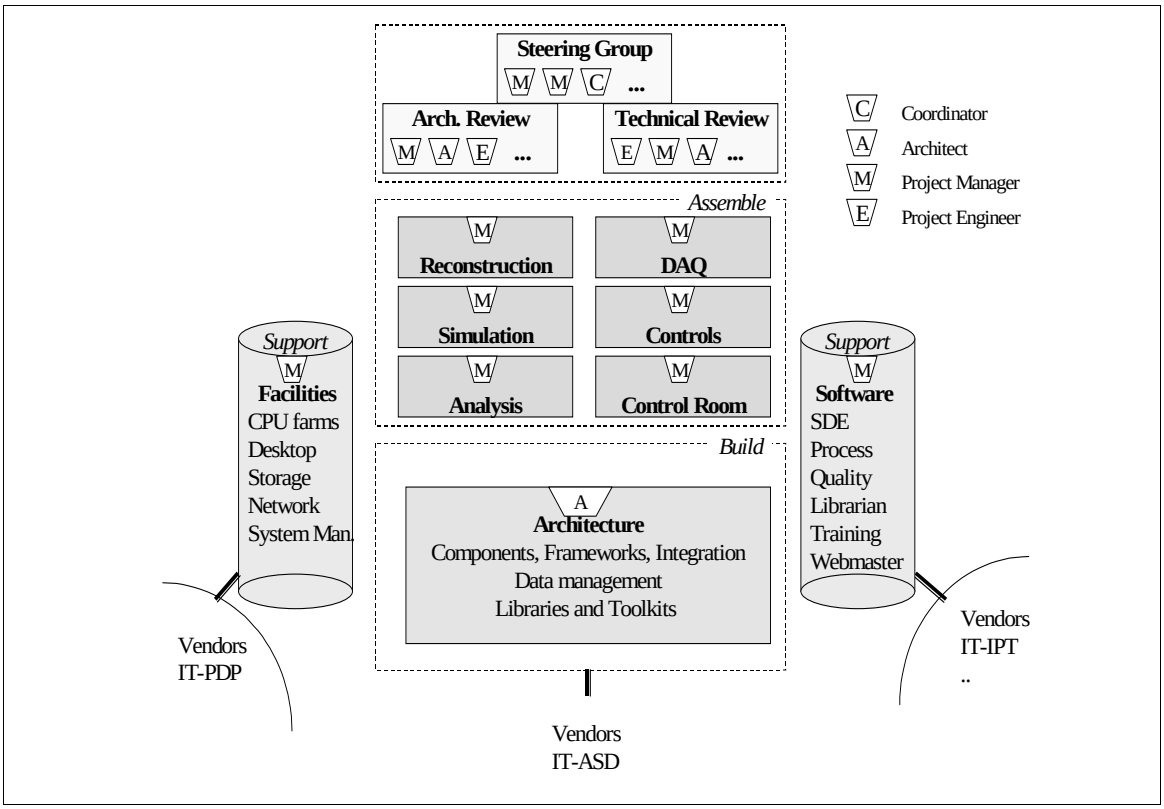

Figure 2 LHCb Computing Project Organisation Structure

1.2 Which parts of the software will physicists write and which parts software engineers?

Our GAUDI architecture identifies a number of software services and components. We expect physicists to contribute to the development of those components comprising physics algorithms and specialist knowledge of the detector. The foundation libraries, frameworks and infrastructure components will be supplied by members of the computing group, who generally have more of a background in software engineering. This complementary view of the software is shown in Figure 4.

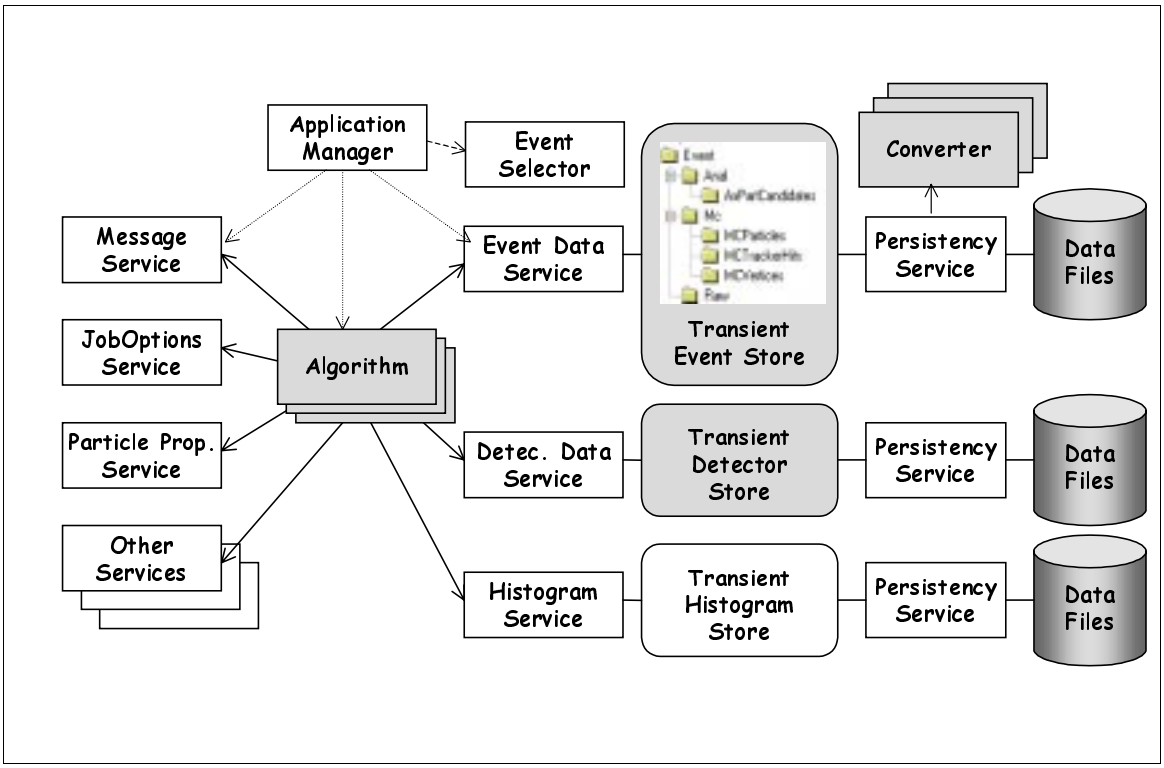

Figure 3 GAUDI architecture showing software services and algorithms that make up our applications

Referring to Figure 3 the physicists will provide Algorithms, the Event model and detector description used to build the specific set of event data processing applications (trigger, reconstruction, analysis, simulation). An analogy can be made to the way the experimental areas are made available: physicists should be given an area where they can install their setup and perform their experiment and the connection to a number of services (power, light, cooling, network, water, etc.) that form part of the infrastructure. You do not expect a physicist to construct the building where he or she will do the experiment. In the same way, you do not expect a physicists to develop all the infrastructure software that he or she will need for the analysis of the data.

.

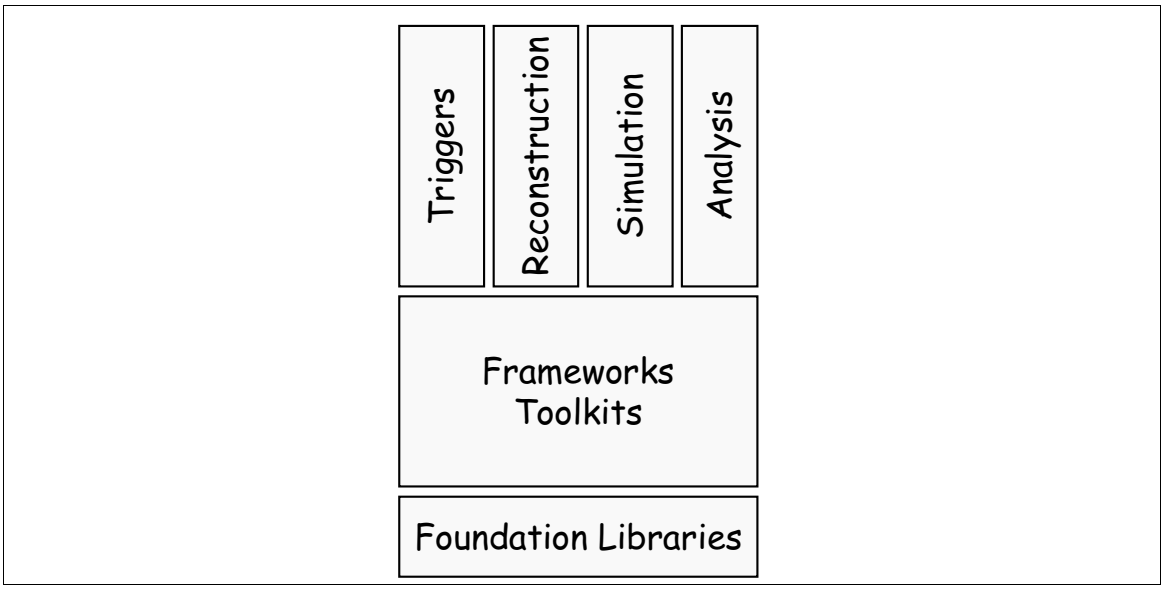

Figure 4 Overall Software structure for Event data processing applications

We expect software engineering skills for developing the infrastructure software (foundation libraries, frameworks, common services, etc.). As far as possible we first look to see if libraries, toolkits and software components already exist that can do the job, and only start to develop ourselves if a component is missing. Third party suppliers might be commercial suppliers, IT departments that provide HEP-specific foundation libraries, or even other experiments.

1.3 How do you stimulate and control contributions from authors spread worldwide?

It is important to delegate responsibility for well-defined pieces of software to groups working remotely. The responsibility for the physics and detector specific software clearly lies with those groups building the detectors. In addition there is software engineering expertise in many of the institutes and so these groups should also contribute to development of the software infrastructure. To date most of the infrastructure software has been built by people based at CERN. We do have a member of the computing team contributing to the development of GAUDI from Orsay and we hope to involve more people as our activity grows.

In practice we have found that it helps significantly if someone has worked closely with the other developers at CERN and then goes back to institute and continues there. There are several examples where this has happened or is happening now.

- Marseilles - main SICb author went there
- NIKHEF - maintain some presence at CERN
- RIO - no contribution yet, but will start soon after CERN fellow takes up position in Rio
- Liverpool - similar situation to Rio

The CERN associates programme provides a useful mechanism for implementing this "policy".

Another important consideration is to make the life of developers as easy as possible. The history of the development of our first FORTRAN based simulation program, SICb, gives some insight into the factors that facilitate worldwide collaboration, in particular the importance of configuration management. The structure of SICb essentially originated from one person and grew into a single monolithic program that performed all 3 data processing steps, simulation, reconstruction and analysis. The software was maintained as a single software package and this proved to be difficult to maintain as others started to contribute and the codebase grew. The maintenance of the codebase was subsequently taken over by a software librarian who reorganised the software into 35 different packages and adopted tools for managing the code repository (CVS) and building releases (CMT). Each package is the responsibility of its main developer, typically someone from the corresponding subdetector group. Each package has its own version number and can be independently released. Each new version of the SICb application is defined in terms of the sum of the individual packages, with their corresponding version numbers. The average LHCb user only follows evolution of the official releases of SICb. More recently SICb has itself been restructured into separate simulation and reconstruction programs. Each developer uses CMT to build private test versions of the program using new versions of his package and is able to perform first tests of the new code he develops. The production team is responsible for making new official releases of SICb and they also perform data quality checks to make sure the results of the simulation have not regressed.

The application of these configuration management procedures has considerably simplified the way in which developers contribute to the overall project. Developers from any institute can commit updates to the central CVS repository. New releases of the application software are made according to a well defined timetable. In addition we are working on an automatic procedure for making builds of the application code (to be used for nightly builds). Modifications made by one developer now have much reduced chance of impacting on the changes made by another. The tools have been quite favourably received and are quite simple to use.

Another development that can simplify the life of outside developers is the use of a bookkeeping tool for recording in detail the data produced in a simulation production run. The data base we are just bringing into operation records the versions of the software used to generate the events, to reconstruct them and to record the version of the geometry of the detector. This database is easily accessible from anywhere via a web-based interface. Grid software providing transparent access to data and cpu resources would be ideal.

In the case of the development of the new OO software a more gradual approach has been taken. The development of the GAUDI architecture and framework started with the appointment of an architect who then assembled a small team (6 people) to work on the specification and implementation of the first version. The team composed the architect, librarian, use-case engineer, and 3 domain specialists (data management, physics analysis). All these people happened to be based at CERN. After release the project was enlarged to tackle new domains and new people were added, including two more people based at CERN and also a colleague from Orsay, who has taken a special role in the visualisation software.

As the majority of the team members are based at CERN communication has been straightforward. In the beginning the initial phase involved a lot of brainstorming. The

GAUDI team met on a daily basis to ensure the project was launched with everyone moving in the same direction. As the basic features and design criteria of the architecture became well established the need for this intense and frequent interaction dropped somewhat. After the first 4 months the frequency of the meetings dropped to twice a week and now they are weekly. It would be very difficult to coordinate this brainstorming phase with a distributed group of people and the rapid progress that has been made can be partially attributed to the fact that team members were sitting close to each other. The increase in scope of the project and the involvement of developers was carefully controlled to take place only after the basic design criteria and development guidelines could be well established.

In contrast to the GAUDI framework, which has essentially been developed by the team based at CERN, the subdetector software is developed by the various sub-detector groups and these are distributed amongst the various LHCb institutes. LHCb has a common training programme, common coding conventions and a library of the seminal books in software engineered, all of which help to engender a common approach to software development.There are several regular activities which help to foster close collaboration. For example, attention is now focusing on helping the subdetector physicists to define subdetector specific event data models, detector descriptions and algorithms and to integrate them within the GAUDI framework. This month (March 2000) a series of reviews of the new subdetector pattern recognition software have started in order to ensure that the concepts and design features are well understood and adopted. Each review involves specialists from the subdetector and from the GAUDI team meeting together. A number of issues common to all subdetectors have been identified (e.g. handling references to MC truth information) and common solutions to solving these issues are being devised. We expect these reviews to be a very important step in managing the interaction between the GAUDI team and the physicists that have responsibility for the physics algorithms. Reviews will be called as appropriate but would typically happen 2 or 3 times per year for each subdetector. In addition we intend to make a yearly major review of the architecture and framework.

More informal monitoring of progress takes place during the software weeks (3 times per year) and during the collaboration weeks (4 times per year). On average there is direct interaction between all members of the collaboration approximately on a monthly basis. All presentations and minutes of meetings are made available through the web pages. So far there has been rather limited use of video and telephone conferencing services.

As a final point we wish to underline that the development of the architecture is extremely important for aiding communication and understanding between software developers. It contains a series of abstractions (services, interfaces, transient and persistent stores to name but a few) and this helps to defines a common vocabulary and style of working. Contributors to the subdetector code know that the framework provides standard services and that they must provide “algorithms” and “converters” to realise their data processing application. As well as fostering communication an architecture helps to foster good design practices.

1.4 What design methodology and design process does the experiment use? Why? How well does this work?

Our software process corresponds well to the USDP approach i.e. it is use-case driven, architecture-centric, iterative and incremental: For example:

- One member of the LHCb software team has responsibility for collecting use-cases (http://lhcb.cern.ch/computing/offline/pdf/gaudiscenarios.pdf). Physicists set the priorities for what should be included in each release.
- Our first step was to appoint an architect and devise an architecture: http://lhcb.cern.ch/computing/Components/html/GaudiMain.html)
- Since starting in fall of 1998 we have made public software releases three times per year. Each release is timed to coincide with an LHCb software week, during which new features are explained to users through presentations and hands-on tutorials. During these weeks priorities for features to be included in the next release are discussed together with physicists using the code.

Intensive use of the GAUDI framework started after the third release in Nov 1999 once most of the required functionality had been included.

We have not yet proscribed a formal documented LHCb software process, nor have we evaluated and selected a particular design tool. Our design efforts have been steered through adoption of a specific training programme that teaches us a standard approach to design (e.g. use of UML as a modelling language), and we have backed this up through the adoption of standard software engineering texts:

- The unified software development process: Jacobson, Booch, Rumbaugh
- The unified modelling language user guide: Jacobson, Booch, Rumbaugh
- Design Patterns: Gamma et al.

An LHCb library is maintained at CERN containing several copies of standard texts and this is well used by all software developers in the collaboration. This has helped to ensure a uniform approach to design.

We have evaluated a number of design tools but have not identified one that is entirely suited to our needs. Rational Rose was evaluated, as it available at CERN. However we found it to be rather complex and to require a steep learning curve. The code generated was found to be unreadable. In fact much effort was being put into making the design in such a way that the code was readable. The general conclusion was that the tool was hindering the progress in developing the software to such an extent that it was dropped. In the absence of any general guidance and support from CERN, we evaluated a number of PC based design tools. These tend to be simpler but are very easy to use. At present we are using a tool called VisualThought which is basically a drawing tool. This is an area which is evolving rapidly, but we do not have the resources to do technology watch. We believe that this is an area where we could benefit from direct support from IT division.

1.5 How have you arrived at your workplan, or work breakdown structure? What planning process has taken place to map out the work to be done in the next two years? Who is responsible for the work breakdown structure and for keeping it up to date?

Information that partly addresses this question is contained in our answer to question 1.6

1.6 What are your milestones in the area of development and how do these interact with other experiment milestones? How will you measure the success of your milestones and evaluate progress in carrying out the work plan?

The major milestones for the experiment as defined to the LHCC are given in Appendix B. They define the principle milestones leading to the definition of the TDR's. At the time of the TDR the technology choices have to be made and a detailed specification of how the appropriate component is to be built must exist. The time leading to the TDR is when experience with different techniques is obtained through prototyping and evaluation studies.

In the case of software the period between now and the start of datataking can be divided into a number of cycles each terminated with a well-defined milestone:

- The first cycle which finishes in the summer of this year represents the first iteration in the production of full scale prototypes in which we expect to demonstrate the appropriateness of our choices for the basic design criteria of the software architecture and the validation of our approach to the organisation of software development activities. We expect to have a new reconstruction program that uses the new framework and allows new OO pattern recognition algorithms to be used in production by the summer of this year. This will allow us to validate physics algorithms that have been re-engineered in OO and whose performance and functionality can be compared to their FORTRAN equivalent. This will be closely followed by an analysis program that uses the GAUDI framework. After this attention will focus on integrating GAUDI with GEANT4 to produce the framework for a new simulation program (timescale not before end of 2000).
- We expect that during the second cycle, which covers the subsequent two year period, sufficient progress will be made to make final technology choices for crucial pieces of the software, such as the persistency solution. At this time we will be in a position to make a TDR for software (July 2002).
- The third cycle will be the period when the engineering of the final software will be made i.e. the software that will eventually be run in production when datataking commences. A large scale test of the functionality, performance and reliability of the software will be made at least one year before data-taking is due to start (summer 2004). This software will be used in conjunction with large scale simulation tests that will also be used to test the distributed data and computing intensive nature of our applications.
- The time between these tests and turn on will be used for integration and commissioning of the complete system and for correcting and problems that have appeared.Efforts will be made to improve performance of cpu intensive pieces of the software. This will be a time of considerable investment in computing resources; cpu, and storage. Time will be needed to get operational experience running large scale compute facilities.

The overall roadmap for the development of LHCb software is illustrated in Figure 5. We proceed in two yearly cycles each terminated by a major review. Decisions on changes to development techniques and major changes in direction can be taken at these reviews. In addition we plan yearly reviews of progress in our software production and we involve

external reviewers to ensure unbiased objective input. Releases of the software proceed in an incremental 4 monthly cycle, and at the same frequency software weeks are organised.

Our current activities are driven by two sets of goals. The first is oriented towards getting physics results from simulation studies that can be used in the preparation of the TDR for each of our detector components. The second is more software oriented and this is to prepare new frameworks for the main data processing applications that use up to date design and coding techniques such that the software can be easily certified and maintained over the lifetime of the experiment. The latter requirement involves adopting a more formal software engineering approach, particularly for critical software components, and in the deployment of OO techniques that are new to many of our collaborators. The challenge is to marry the two sets of goals by carefully preparing a migration strategy that allow physics studies to proceed with little interference, whilst at the same time to encourage and allow new software to be developed so as not to add to the legacy code. This migration strategy has been the subject of much debate and discussion within the collaboration at the weekly computing meeting and in the collaboration meeting where it was finally approved.

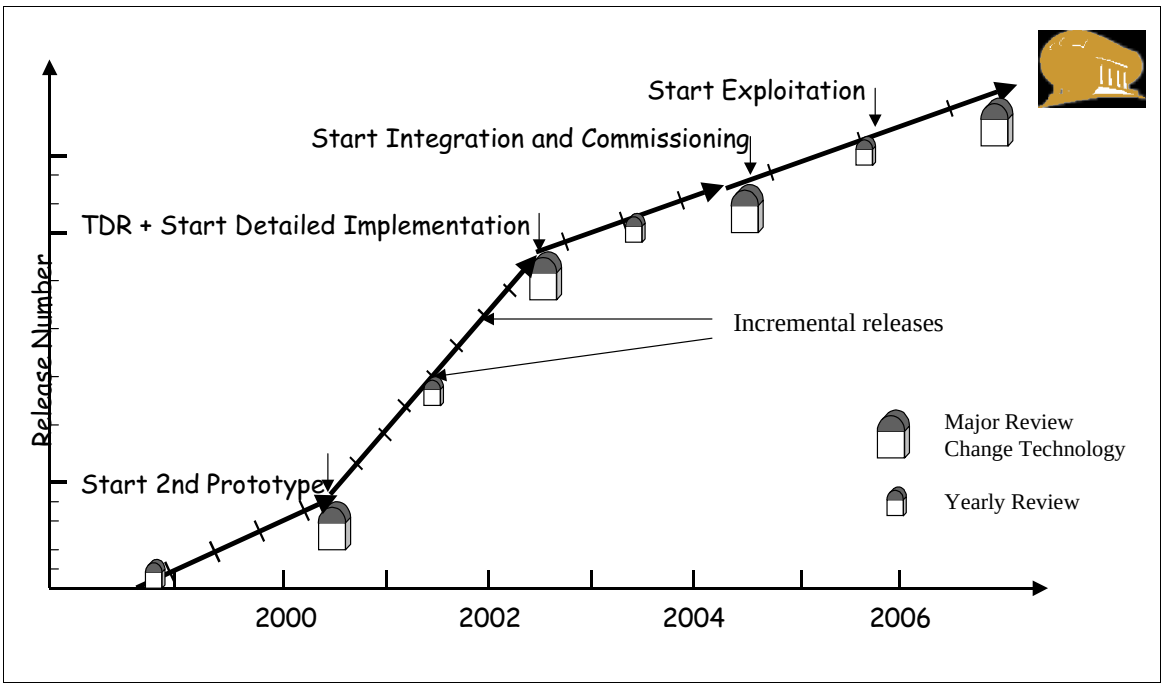

Figure 5 LHCb Software Roadmap

Clearly the production of the TDRs for the detectors places stringent requirements on the software used in the detector optimisation studies. At present the production simulation uses the existing FORTRAN based software. The migration of the framework of the reconstruction program to use GAUDI is almost complete. However the migration of the individual physics algorithms to use the new data model, detector description and other GAUDI services is still in progress. The timescales for each subdetector to complete this work are still being discussed. This issue will be discussed at the next software workshop (April 5-7) when we hope to have more detailed information.

The success of the milestones can be measured in terms of the existence of the deliverables associated with each at the appropriate time. Where possible the attributes of each deliverable will be measured and compared to the requirements e.g. performance. The software will be continuously be used in production (simulation) to get physics results. It will be exposed to users. The timeliness of the delivery is easy to measure. Project plans will be produced and used to track progress.

Concerning the development of GAUDI, the project leader has kept a detailed project plan since the start of the project describing ongoing tasks corresponding to each release of the framework (Appendix A). A more detailed day to day joblist is also maintained and reviewed at the weekly meeting. This list is maintained on the GAUDI web pages.

We have also developed a detailed plan for the migration of our software from SICb to use the new GAUDI framework. This joblist was presented and endorsed by the collaboration at a collaboration meeting. Progress is tracked at the weekly meeting, and on almost a daily basis via personal contact. Responsibility for the migration is shared between 3 people for the three steps that have to be undertaken:

- Step 1 - splitting of SICb into simulation and reconstruction parts. This has been completed
- Step 2 - wrapping of FORTRAN code such that it can be called from GAUDI. Creation of the first version of the new reconstruction program BRUNEL (hybrid phase)
- Step 3 - gradual replacement of FORTRAN with OO algorithms that use GAUDI services.

The development of BRUNEL is just starting and will be managed in the same way as GAUDI, with a project leader who has responsibility for obtaining and integrating software components from all subdetector developers. Attention will be given to planning tasks, identifying risks, understanding critical paths, and coordinating efforts so that timescales and deliverables are respected.

1.7 Which areas of planning and work are the highest areas of risk, in that lateness or poor quality will have far reaching affects? What is being done to mitigate these risks?

In general we believe that today, considering the experiment as a whole, software is not on the critical path. Most attention in the collaboration is devoted towards making the right technology choices for the detectors and optimising the overall design. However if we just address software then there are a number of issues which require strategic thinking.

One of the biggest challenges we face on LHC experiments is the management of the very large and complex data sets that will be produced. The complexity of the data access patterns and the requirements on caching and replication of data imply that sophisticated software will be needed to manage and access the data. It is important that we have complete confidence in the choice of the software used to for data storage management and have sufficient control over it that we can ensure it meets all our requirements. Efforts to provide a solution to data have concentrated on looking for a commercial solution from the ODBMS market. Current trends in the evolution of the market and experience of some of the technical

limitations of the product have been grounds for legitimate concern. The alternative of providing a home grown solution would also be costly in terms of development effort.

If a home-grown solution with ODBMS-like features is to be developed then there will need to be a significant investment in experienced manpower and sufficient time allocated. This is an issue which requires cooperation and agreement between all experiments to find an adequate solution that will mitigate the risk, if necessary by pooling resources. An open debate on this issue is needed rather soon. Another aspect that can help to mitigate the risk is to avoid making the software dependent on a particular persistency solution. One of the basic design decisions we have taken in devising the GAUDI architecture is to separate the transient and persistent representations of the data. This means that algorithmic code has no notion of how the data are physically stored, such that and particular persistency solution can be rather easily replaced by another. At present we make use of two persistency solutions, ZEBRA, which is used for legacy FORTRAN data, and ROOT.

Stringent quality requirements must be in place for certain parts of the code, such as the high level trigger algorithms. Quality can only be ensured if correct procedures are introduced to the development of the software. These include design reviews and code inspections. Data quality checks will be introduced to verify the correct functioning of the code on test data samples. These checks will be applied on every new version of the software to ensure it hasn't regressed.

Other areas of risk arise form the change to using new software techniques. The business of producing software is becoming far more technical and requires more skills. It requires an investment in effort and time that not every developer is prepared to make. Heavy use of training programmes and mentoring is required to help people over the technology hurdles.

1.8 What is the plan for training? What are the various types of training required - design? use of tools? C++, other? What have you learned so far about the successes and failures of training programs and what do you intend to do in the next two years? Do you expect that any of this be in common with other experiments? What role do you expect CERN IT to play?

What is the plan for training? What are the various types of training required - design? use of tools? C++, other?

A decision was taken to use C++ as the official LHCb programming language, at least for the first iteration of the software life cycle. This decision may be changed subject to a major review at the end of 2000. Nearly all LHCb physicists programming in C++ have followed the course on C++ for Physicists given by Paul Kunz.

In addition a 5 day course in OO analysis and design and hands on programming in C++ was devised together with the consultant, John Deacon, especially for LHCb. This course is also now offered in this form through the CERN technical training programme

(http://www.cern.ch/Training/tech/TE_main_e.htm). In total more than 50 LHCb software developers have now followed this course.

Feedback from participants included the following comments “very useful and complementary”. In general they appear to be well accepted as very useful introductions to the new OO techniques. The difficulty is to provide follow-up help and advice once the developer is working on his project. For this we rely on mentoring from more experienced people in the group who can give practical advice as and when needed. For this reason it is better to start the software activity in a controlled way, starting from small teams with one or two experienced OO programmers who can act as mentors to those less experienced. We expect the review procedures to also play an important role in providing this mentoring.

In conjunction with the course designs have been made using the standard UML notation and further self-training has been possible through the adoption of standard software engineering texts, such as:

- The unified software development process: Jacobson, Booch, Rumbaugh
- The unified modelling language user guide: Jacobson, Booch, Rumbaugh
- Design Patterns: Gamma et al.

An LHCb library is maintained at CERN containing several copies of standard texts and this is well used by all software developers in the collaboration. This has helped to ensure a uniform approach to design.

What have you learned so far about the successes and failures of training programs and what do you intend to do in the next two years?

It is not sufficient just to learn the programming language. Any physicist wishing to do serious OO development, such as designing algorithms or the event model, needs to know the basics on object orientation. All LHCb physicists working on these topics have been encouraged to follow the OO A&D course. Positive feedback from participants has been helped to ensure that a large fraction of our physicist developers have attended this course.

To be successful the training must be timely, new skills need to be put into practice immediately otherwise the benefits of the training are lost. It may even be preferable to attend a course shortly after having started a project and to use the occasion to ask questions that have already been formulated in the participants minds.

Several collaborators have attended other courses on specific products that we are using. Examples include Objectivity for object persistency and PVSS for the Controls System. Again these courses are organised through the CERN technical training programme. A major involvement in such courses is waiting on final decisions on the LHCb persistency solution and the choice of a commercial SCADA system. We would expect that over the next couple of years more specialised training in specific products will be needed. One example is GEANT4, which is described in more detail below. Another example may be in the use of commercial design tools.

In addition we have to devise high quality training material for our own software, such as GAUDI. What exists so far will need to be improved. More examples are needed to help LHCb developers using the framework. We are minded to develop workbooks containing

tutorial-style material that can be offered to new collaborators and help to accelerate their use of, and contributions to, LHCb software. These workbooks could well be along the lines of the BABAR offline workbook (http://www.slac.stanford.edu/BFROOT/www/doc/workbook/workbook.html). We would intend to have three such workbooks,

- one oriented to users of the software,
- one oriented to software developers which would contain basic material on the engineering approach, how to use the code repository etc.
- one on managing LHCb projects, which will explain how to setup a project.

Now is a timely moment to launch this activity, we are only have to find the effort so that it can be started.

Do you expect that any of this be in common with other experiments?

Training courses have been formulated with the help of CERN's technical training department. It so happens that over the last four years someone from LHCb has been on the committee that defines the CERN software training program and so we have had an opportunity to influence its contents. Courses offered through this programme are made available to anyone and LHCb would intend to benefit from any courses offered by the training department. For example, the OO A&D course that was set up for us has been added to the training programme and has been taken and adapted by other experiments to fulfil their specific training needs. Courses in Controls are organised through the CERN LHC Joint Controls Project (JCOP) and are attended by representatives from all LHC experiments.

What role do you expect CERN IT to play?

We expect CERN/IT to play a role in providing specific training material on software developed by IT and in organising training courses on the commercial software selected for the main CERN libraries (such as Objectivity). Training in Objectivity already exists. In addition we foresee a special need for GEANT4 training material. Such material does exist, having been produced by GEANT4 members, largely those also working in experimental collaborations. It would be very useful if this material could be made available to our collaboration as we do not have many collaborators who have contributed directly to the GEANT4 software and do not have experience in its use. This could be a very useful role for the IT departments to collect this material and help to organise training.

1.9 How will technology choices for languages, tools, database products, etc. be made? What provisions are being made for rapidly changing technology?

In one hand, we would like to track the changes in the technology and profit from the possible benefits of new technologies immediately. On the other hand, we are also aware that we need

to have a periods of a certain stability to allow the development of specific software in a smooth manner. The two wishes are contradictory and the compromise we have found is to fix the technologies (languages, tools, persistency, etc.) for a periods of 2 years. During the general reviews of the computing project (scheduled at 2 year period) we could be fixing the choices for the next period.

1.10 What plans do you have for the long-term support of your software?

A careful approach to configuration management is needed. All software is managed by our software librarian using a CVS code repository. Physical design guidelines are followed to help minimise problems at compile and link-time. We also use a release tool called CMT for building libraries etc.

We expect to have to deal with a high turn over of collaborators given the long time scales involved. Maintenance can only be eased by having a software process that addresses this issue. Each step in the development process needs to be defined and high quality material needs to be produced documenting requirements, designs, code, and test procedures. Where possible (semi) automatic procedures should be used to produce this material to ensure that it is kept up-to-date and this implies extensive use of software tools.

We have put some considerable effort into documenting GAUDI. Software reference manuals are produced automatically using a tool called ObjectOutline (http://lhcb.cern.ch/computing/Support/html/objectoutline.htm).

Reviews generate a lot of useful material, documentation is produced before-hand and the results of the review must also be documented. See ***Architecture Review Documents*** on http://lhcb.cern.ch/computing/Components/html/GaudiMain.html.

Training material needs to be produced for each step. Workbooks seem to offer the most convenient format for material to help new collaborators.

1.11 What quality assurance and control mechanisms are being put in place, and in which stages of the design, implementation and testing processes?

At present we do not have sufficient manpower resources to put in place a proper software quality process. Areas where we have made some progress are

- design reviews - we have made reviews of GAUDI and have started a series of reviews on subdetector software (calorimetry, tracking and RICH)
- coding conventions - we have established a coding conventions guidelines document. We have not yet put in production the automatic checking of code checked into the repository. We are awaiting the outcome of a project started by IT/API group to put in place a formal procedure.

- data quality monitoring - following each new release of SICb the production team checks the output against a standard set of histograms that represent the understood behaviour of the program. This represents a simple regression test. We expect to apply such data quality checks on all future versions of our data processing software.

1.12 What decisions on software technology and implementation choices have to be taken in the future and when do you plan to take them?

Technology choices will be reviewed every 2 years. For example we are currently using UML as a design notation and C++ as an implementation language. We are also putting some effort (work of a technical student) to evaluate other languages (Java) and at the next major review we will debate the advantages of introducing Java where it is the most appropriate language.

Choices still need to be made for some of the basic toolkits used for implementing framework services. The example of the persistency service has already been mentioned. For the simulation toolkit we are starting to get experience with GEANT4.

The technologies to be used for the development of the software to be run in 2005 will be defined in the software TDR (expected 07/2002).

1.13 What is the required number of people contributing to the software, what is the break-up between physicists and software engineers? What is the evolution over time to meet the milestones (manpower profile)?

The tasks to be done developing and managing different parts of the software are shown in the following tables.

Table 1 Manpower requirements for software framework and support

Job	Description	Type	FTE need/have
Project Leader	Project Management	Eng	0.5 / 0.5
Software architect	responsible for software architecture and framework	Eng	0.5 / 0,5
Event Data Model	Generic model for event data for all data processing stages (raw, reconstruction and analysis, event tags etc)	Eng	1 / 1
Detector Description	Responsible for all data related to the detector, geometry, calibration i.e. conditions database	Eng / Phys	1 / 1 (student)
Data storage management	Responsible for the persistency solution ODBMS, use of mass storage	Eng	1 / 1
Bookkeeping	Manage production databases and related tools, RDBMS applications	Eng	1 / 0.5
Visualisation	Graphical data representation, graphics packages, GUI, interactive tools, event display framework	Eng	1 / 0.5
General Services	Job options, messaging, application manager	Eng	1 / 0.5
Simulation Framework	Coordinate development of simulation program. A second person (physicist) is needed to manage the physics generators.	Phys / Eng	2 / 1
Reconstruction Framework	Coordinate development of reconstruction program	Phys / Eng	1 / 0.5
L2 / L3 frameworks	Coordinate development of frameworks for Level 2 and Level 3 trigger algorithms, ensure quality control	Phys / Eng	1 / 0
Analysis Framework	Coordinate development of analysis framework	Phys / Eng	1 / 0

Table 2 Manpower needs for software support activities

Job	Description	Type	FTE need/have
Process Engineer	Collect use cases, apply USDP procedures, manage software tools, manage training	Eng	1/0
Librarian	Manage code repository and release procedures	Eng	1/1
Quality Manager	Organise design reviews, code inspections, test procedures, data quality monitoring	Eng	1/0
Collaboration Tools	Manage the collaboration database, the LHCB web, videoconferencing facilities etc	Eng	1/0
Production Manager	Manage productions of simulated datasets	Eng	1/1

The effort involved in the development of subdetector software can be described as follows:

- **Muon detector**: At present the FORTRAN (SICb) Muon software is maintained / developed by Paul Colrain, Gloria Corti and Andrei Tsaregorodtsev with some assistance from 3 or 4 people from Rome and Rio. Present OO software is being developed by Miriam and Paul. Future software development, both in FORTAN and OO(C++), will be separated into 2 areas:

    1. 1. Muon Detector (Detector Description Database, Digitization and Muon ID)
    1. 2. Muon Level 0 Trigger (Online and Offline)

  There will of course be close collaboration between the two groups. The development of the first part will be undertaken by the Rio, Rome and CERN groups. Within the Rio group there will be (from July) 2 physicists and 1 OO Software Engineer working on this. There is strong interest from two physicists from Rome and maybe 1 from CERN to contribute. I believe this will be enough and expect a flat profile.

  The Marseille group will take care of the Trigger Software development. There will be of the order of 2 physicists working on this.

- **Tracking system**:

  Inner Tracker Simulations + Digitizations: 0.5 fte

  Outer Tracker Simulations + Digitizations: 0.5 fte

  Detector level Reconstructions:

  Inner tracker specific: Si clustering or other 0.5 fte

  Outer tracker specific: r-t drift relation, l/r ambiguity etc. 0.5 fte

  Pattern Recognition:

  Trackseeding: 1 fte

  Trackfollowing: 1 fte

  Track refit: 1 fte

  Fast Version of tracking: 0.5 fte

  Alignments: 0.5 fte

  In total it adds up to 6 fte's, all physicists (mainly: grad stud.+postdoc)

  Currently, we have roughly the required manpower (including 2 fte's currently starting up).

- **Vertex detector (VELO)**:

  Experience so far is that 2 FTEs have gone into development of geometry, simulation and reconstruction 2 FTEs and in the test beam 3 software engineers and 4 physicists working part time.

  For the future we estimate needing one dedicated software engineer for the VELO for the testbeam code and the GAUDI simulation/reconstruction code, and at least 2 physicists who are OO experts to write and maintain the reconstruction code.

- **Trigger**:

  L0: there are ~ 3-4 physicists working on it now.

  L1: 2-3 physicist working on it now.

  L2-3: nobody for now, need 1-2 starting this year.

  Experience has shown that in the beginning we need we need at least a couple of soft-engineers extra, let us say 1 dedicated to L0 and 1 dedicated to L1, to bring the necessary OO experience. We need extra physicists too since a reasonable fraction of their time will be spent at interacting with the engineers. A starting fellow/student needs training+experience in C++, which is not as easily acquired as with FORTRAN, and hence is less productive in the short period he/ she is with LHCb.

- **Calorimeter**

  1 person for overall design and project management - 1 FTE

  detector description, GEANT simulation and digitisation requires 1 person for each component of the calorimeter (ECAL< HCAL and preshower) total 3 FTEs

  For reconstruction 1 person needed for clustering and one needed for particle identification total 3 FTEs.

  Analysis of testbeam software involves 3 people - total 1.5 FTEs.

  The profile of all people is physicist.

- **RICH**: waiting input

We expect the manpower needs to be fairly flat between now and 2005.

1.14 What is the recruiting model for manpower?

Recent priorities at CERN have been to consolidate general support to physicists by recruiting a computer scientist to help with system management, trouble shooting problems on desktop machines and on facilities used by collaborators at CERN. This must be supplemented by securing funds for obtaining services from the CERN desktop contract.

The highest priority for the next position is to hire a physicist with software experience to work firstly on studies of the impact of background radiation. This person would also work on the analysis framework as time permits.

A significant amount of effort is to be acquired through the students and fellows programme at CERN. Our first priority is to consolidate the development of the online system by acquiring an Applied Fellow, which we hope to do in the summer. These people will have an applied background i.e. we look to hire software engineers.

1.15 How do you plan to provide working software and do development at the same time? How do you plan to transition from existing software to final production software?

The simulation program used in the detector optimization and physics studies to date is written in FORTRAN and uses GEANT3 as a simulation toolkit and ZEBRA as a data management system. At the time of the Technical Proposal we took the decision to engineer from scratch new data processing software using the OO approach and appropriate programming tools and techniques. We therefore had to cope with a shift of paradigm, from structured to OO, and the use of new programming techniques and languages, C++ instead of FORTRAN. In migrating from the old to the new we wanted to preserve as much as possible of the investment in knowledge about the detector and the algorithms used to extract the physics that went into the development of the original code. We also wanted to ensure that those responsible for the original software continued their involvement and maintained their

responsibilities in the new software activity, and in this way we intended to avoid fragmentation of the software effort.

There were a number of practical issues that had to be considered and which motivated the rapid introduction of the migration strategy. Firstly there was an urgent need to be able to run new tracking pattern recognition algorithms, which had been written in C++, with standard FORTRAN algorithms in production and in time to produce useful results for the detector TDR's. There was also a practical software goal, namely to allow software developers to become familiar with GAUDI and to encourage the development of new software algorithms in C++.

We therefore produced a detailed plan for the migration to the final software, proceeding in a number of well-defined steps and according to an agreed timetable. A number of different strategies were considered and these are shown in Figure 6. The first would have been to have two parallel developments, presumably with different people working on both lines, with the intention of eventually merging the two. This was soon discarded on the basis it did not meet the goals outlined above. The second possibility would be to rapidly translate existing FORTRAN algorithms into C++ and to integrate them with the GAUDI framework. This would have been an attractive option if it could have been done quickly and efficiently. The third option, and the one eventually chosen, was to take the existing algorithms and wrap them so that they could be called from GAUDI. This strategy was presented to the collaboration in November 1999 and subsequently endorsed by the collaboration and LHCb management.

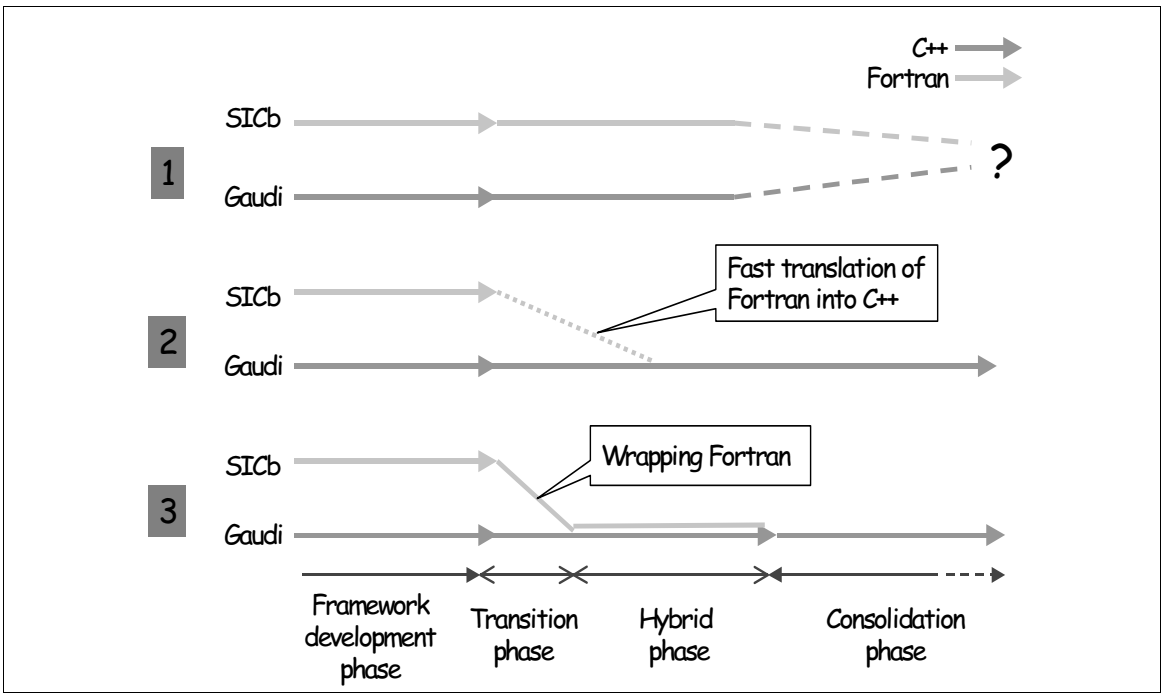

Figure 6 Possible Migration Strategies

We then decided to start with migration of reconstruction and analysis code and to leave the simulation code until the first part had been completed. This was due to the fact that the

simulation is to be based on GEANT4 and we still have to gain experience using the GEANT4 toolkit. The migration is now proceeding in three steps

- Step 1 in the procedure involves restructuring the existing FORTRAN into its simulation (called SICbMC) and reconstruction (SICbREC) components.
- Step 2 is to wrap digitization and reconstruction Fortran modules in GAUDI. The nett result is a new reconstruction program which we call BRUNEL. Once this procedure is completed (transition phase) then the GAUDI based application becomes the production version and the original program SICbREC can be retired from service.
- In Step 3 the FORTRAN algorithms are gradually replaced one by one with new OO algorithms that use all services and features of the OO framework (event model, detector description etc.). This is the hybrid phase and the aim is to keep it as short as possible as during this time FORTRAN and OO representations of components, such as the detector description will have to be maintained. start replacing FORTRAN modules with C++ equivalent.

The current status is that Step 1 has been completed and Step 2 is due to be completed by the end of March 2000. The time required to step 3 will depend on each subdetector group and we hope to have better indications of their plans in our next software week (April 5-7, 2000).

1.16 Given that the support from CERN/IT is limited, how do you identify the areas where you would most like to see strong CERN/IT involvement and support? What are the arguments for central CERN support?

We would like to see (not in priority order):

- guidelines and support for “organisation, methods and tools”, for documentation and information management.
- technology tracking on tools, manage company contacts, handling licence agreements etc.
- Support for foundation libraries, GUI, MINUIT, particle properties, persistency, data management,...
- items not strictly software, such as tools for control and operation of compute farms, management of grid computing etc.
- in the area of online, we rely on the controls group to supply software via the JCOP project

Arguments for are based on the optimisation of resources, centralisation of contacts with industry etc. However development of software for the collaborations needs to be driven by an experiment’s needs. The model for managing IT based software projects needs revision to ensure that whatever is produced can and will be used by the experiments. The procedure setting priorities and for taking decisions is not clear. The only forum for interaction is via a fortnightly meeting where the format is information exchange. There is no clear procedure for defining the tasks and strategy for the development.

2 Architecture, Data Model and program infrastructure

2.1 What requirements, current and future, have you identified for your software architecture or 'framework'?

A set of scenarios (use-cases) were collected at the time the architecture was designed (see [1]) to capture the main system requirements. More detailed functional requirements are being collected in the subsequent iterations of the framework.

We use the term scenario quite loosely and synonymously with the term use-case. We have used scenarios as a way to deduce the system requirements. We collected the scenarios by interviewing directly some potential users. In the course of our discussions we came across things which "it would be nice to have", but were not really formulated either in terms of scenarios nor as requirements, we have grouped these together under the term: Desirables.

We have grouped the scenarios according to the type of individual who is most affected by or interested by that particular usage. Of course, often several different types of user are involved. In places we refer to "users". Usually this term refers to someone who does not write code, but only "uses" the software. In our environment such people do not exist since everybody writes code for one reason or another. We have identified the main groups of people who will work with the framework, as follows: physicist users, physicist developers, data production managers, framework developers. Besides the "users" we also tried to take into account the expectations from other "stakeholders" that are not necessarily users of the framework i.e. managers, librarians, etc. Their expectations are more in terms of qualities of the software than functionality. For example: maintenability, adaptability, flexibility, etc.

2.2 How will it evolve? How will it meet demands for parallel processing?, distributed processing?, security and authentication needs? Language evolution?

How will it evolve?

It has never been our intention to produce a complete set of requirements. The traditional "water-fall" model does not work [2]. It has always been our assumption that we do not know all the requirements at start-up and that we will be discovering them during the subsequent development iterations. Therefore requirements will evolve and we are prepared for that.

We are convinced that by using the USDP software development process [3] (use-case driven, architecture-centric, iterative and incremental) we can handle easily the evolution of the requirements. In particular, having defined an architecture resilient to changes is our best bet.

We started by having a rough idea of the system requirements (use-cases and scenarios) enough for defining the architecture. In each development iteration of the framework (release every 3 or 4 months coinciding with a software week) we collect the feedback from users and compose a prioritized list of

new wishes or wanted functionality. Therefore the requirements are evolving continuously. Sometimes in this evolution we are even formulating contradicting requirements. This only shows that our level of understanding of the experiment and technologies involved is maturing with time.

Our philosophy for the software development has been “start very simple and add later the complication if needed”. If we start with very complex solutions we will fail in their adoption by the end-users.

How will it meet demands for parallel processing?

No real parallel processing. What we plan is to use the “trivial” parallelism exploiting the nature of our event data where each event is processed in a different loosely coupled processor (processor farms). We have taken into account for the design of the architecture that applications based on that architecture will also be the high level triggers (Level-2 and Level-3). These software triggers will be run in massive processor farms (~2000 CPU boxes). Therefore, we will need to evaluate how the software behaves in this very different environment (on-line, parallel processing, etc.). We certainly will need to add some more functionality to meet the stringent demands.

Distributed processing?

We plan to add distributed processing capabilities to the framework at later stage. The idea is to be able to distribute the GAUDI services over the network. This is technically possible since only the abstract interfaces are visible and accesible from the services. For example, we could use CORBA or DCOM to map easily these services. This should not be complicated since our interface model is similar to DCOM. The non trivial problem left is distributing GAUDI data objects (event and detector data) if performance is required.

Security and authentication needs?

Again this is something we have in mind as a possible need but we do not want to complicate the first releases of the software. We will implement it later if we are convinced that is needed, once we have definined the level of security required.

Language evolution?

We are currently implementing the architecture (framework) using C++. But we did foresee as scenario during the development of the architecture a possible change of programming language. In particular, we considered very seriously a later evolution to Java. For that reason, we did design the architecture with “Java in mind”. This means that we use only constructions that will be easily translatable into Java. For example we avoid using multiple inheritance, the interfaces we define behave as Java interfaces, etc.

We are currently doing an evaluation of Java. With that we evaluation would like to gather as much information as possible for an eventual migration to Java at later date. In practice, we have translated

the GAUDI framework into Java and we plan to write a typical physics algorithm to evaluate the performance and asses the qualities of this approach.

2.3 Do you believe that your requirements for framework, software build and release process and data persistency are distinct and different from the other experiments?

The quick answer to this question is NO. We believe the requirements for the framework are the same. In fact, we have less demanding requirements in terms of performance, data throughput and data volumes than the other general purpose LHC experiments. In terms of qualities, we have very similar demands. We also believe that sharing the same implementation, or parts part of it can be very beneficial to our experiment. Certainly we can share the same Framework with other experiments, the Framework being a set of abstract interfaces and a number of common infrastructure services. Of course, we will have a different “Event Model” and different “Algorithms” but many of the infrastructure services can be the same.

The problems we are encountering for the software build and release process are very similar to other experiments. In fact, LHCb having started later, we are re-discovering problems that other experiments have already encountered. The requirements for tools helping us in building and releasing software are also very similar, besides small additional requirements based on our past experiences. We are convinced that the tools can also be common. This does not mean that we all should use a single tool that fits the union all possible requirements and constraints. This ideal tool probably does not exists. The other extreme is that each experiment uses its own specific tool. Sure something in between is the right approach.

The requirements for data persistency are also not different between the experiments. The problem is that even in a single experiment you may have quite different requirements for the different nature of data that you want to make persistent (event data, event catalogues, calibration data, bookkeeping data, etc.). We are convinced that a single persistency solution which fits all type of data is not realistic. More over, we are concerned with the long term changes or evolution in the data persistency technology. One of the scenarios we took into account when designing the architecture was to handle the case in which the persistency solution would have to be changed during the lifetime of the experiment.

2.4 How is your data model defined? What do you mean by your data model?

In GAUDI we have decided to separate data from algorithms. For example, we are thinking of having “hits” and “tracks” as basic data objects and to have the algorithms that manipulate these data objects encapsulated in different objects such as “track_fitter” or “cluster_finder”. The methods in the data objects will be limited to manipulations of internal data members. An algorithm will, in general, process data objects of some type and produce new data objects of a different type. For example, the cluster finder algorithm produces cluster objects from raw data objects.

We envisage three major categories of data objects. There will be the event data which is the data obtained from particle collisions and its subsequent refinements (raw data, reconstructed data, analysis

data, etc.). Then, there will be detector data which is all the data needed to describe and qualify the detecting apparatus in order to interpret the event data (structure, geometry, calibration, alignment, environmental parameters, etc.). And finally, we will have statistical data which will be the result of some processing applied to a set of events (histograms, n-tuples, etc.).

What do you mean by your data model?

What we mean as **data model** is the structure of the transient data made available to the algorithms.

The organization of the data in the transient data store will be a tree of data objects. This structure resembles very closely a typical file system with files and directory files, where directory files may contain also some data attributes. This is shown in Figure 7. Each data object can potentially be a node of the tree and in addition also contain its own data members or properties. For example, the Event object is the root node for all the event data and has a set of properties, e.g. event number, event time, event type, etc.

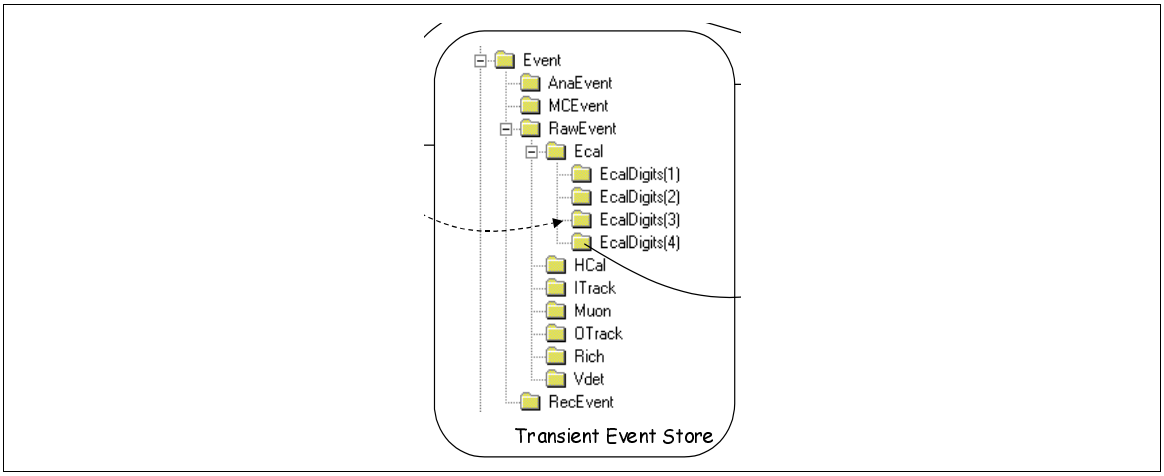

Figure 7 Transient data store

Any object in the data store needs to be uniquely identified. As in the case of the file system, the identification (i.e. file name) is unique at the level of its container. The “full path” identification that uniquely identifies the object in the store is then made by appending the identifiers (names) of all the ancestor nodes with its own identifier.

We are aware that most of the event data objects will be very tiny data objects. For example, for each event we will have thousands of hit objects each of which will be a few bytes long. We do not want these tiny objects to incur a big overhead when being managed in the transient data store (the same arguments applies for the persistent store). Therefore, we foresee objects in the store that will be normal identifiable objects and in addition be containers of small objects. These small objects themselves are not identifiable directly, but rather by containment.

Having this strong hierarchical structure between data objects (aggregation) does not preclude other kinds of relationships between the different objects, e.g. we can have a hierarchy consisting of the event root, raw event, a number of sets containing hits and a number of sets containing tracks. On top of this hierarchy we can have a relationship between hits and tracks.

How is your data model defined?

The identifiable objects are typically a container (templated container that inherits from a basic “DataObject”) of objects of a given C++ class defined in a header file. In principle, any class can be a contained object. The only restriction is the use “Smart References” to implement relationships to other objects. This allow us to implement “load on demand” of the referenced object.

2.5 How does this interact with the data persistency mechanism? Language choices? Analysis tools?

A main feature of GAUDI is the separation of the persistent data from the transient data for all types of data e.g. event, detector description, histograms, etc. We think that physics algorithms should not use directly the data objects in the persistency store but instead use pure transient objects. Moreover neither type of object should know about the other. There are several reasons for that choice:

- The majority of the physics related code will be independent of the technology we use for object persistency. In fact, we have foreseen to change from the current technology (Zebra) to an ODBMS technology preserving as much as possible the investment in terms of newly developed C++ code.
- The optimization criteria for persistent and transient storage are very different. In the persistent world you want to optimize I/O performance, data size, avoid data duplication to avoid inconsistencies, etc. On the other hand, in the transient world you want to optimize execution performance, ease of use, etc. Additionally you can afford data duplication if that helps in the performance and ease of use.
- To plug existing external components into our architecture we will have to interface them to our data. If we interface them to our transient data model, then the investment can be reused in many different types of applications requiring or not requiring persistency. In particular, the transient data can be used as a bridge between two independent components. For example the conversion of the geometry objects from a given representation the the G4 representation.

Having made this choice on separating the transient from persistent representation, the “data model” we are defining is completely independent from the persistency technology we are using or going to use in the future.

Language choices?

It is clear that the current way we define our data model (i.e. using a C++ class definition) is not the best way when trying to mix languages or evolving from one language to another. A programming language independent object definition language would be much better. We are not aware of the existence of such a language, unbounded to persistency or distributed computing solutions.

Analysis tools?

It is a fact that some of the popular analysis tools are bounded to a specific persistency solution. This is a problem. If these analysis tools where designed in an another way we could easily plug other persistency solutions while keeping the data analysis functionality. The current solution we have is that we produce statistical data (histograms, n-tuples) in the form that analysis tools require. This is not the ideal solution because our event data model is not known to the analysis tool.

2.6 What infrastructure will be used to assure that all conditions, parameters and code which were used to create a data object are codified and known?

What we are currently doing with the FORTRAN software and the new C++ software in terms of codifying the conditions, parameters and code used to create a data object is based in the **configuration management** system and **bookkeeping database**. We have plans to enhance the system by incorporating into the GAUDI framework an extra package on the lines of the Run Control Parameter (RCP) of Fermilab. With this, we could track “all” parameters used by “all” algorithms involved in the creation of a data object.

Configuration Management

The version of the code and some input data files (particle decays table, detector geometry, etc.) used to produce new data is controlled using a set of configuration management tools. Currently the tools we use to support configuration management are: CVS to manage the code repository and CMT for managing the build and release of versions of the software. The current practices in LHCb are described in [4].

Bookkeeping Database

The bookkeeping database contains the list of all data sets (currently only monte carlo data but in the future also experimental data) available to physicists of the collaboration. Together with the identification and the physical location (id, tape number, etc.) of each data set there are a number of parameters that fully qualify the data set. The main purpose of this extra information is to be able to perform sophisticated queries based on them to select the data required by the physics analysis. In addition, some of the parameters are there to codify how the data sets have been produced (program version, database versions, parameters, etc.). The current implementation of the bookkeeping database is based on ORACLE. The database server is managed by the CERN IT-DB group and accessible by clients worldwide.

2.7 What mechanisms are you using to assure that the core infrastructure components of the software have broad experiment input, validation and testing?

The mechanisms we use to ensure the input from experiment collaborators to the framework and infrastructure components are listed here:

- Participation in defining “use-cases”. Subdetectors physicists provided requirements in terms of “use-cases” or “scenarios”. Their input was obtained by personal interviews.
- Weekly computing meetings. The computing meetings are a forum of discussion of requirements, feedback and progress reports between the sub-detector groups and the core computing group.
- Incremental releases. A new version of the framework and basic infrastructure is released 3 or 4 times a year with added functionality in each release. In this way the “users” have an opportunity to use the new functionality and provide feedback. This feedback is used to determine what needs to be changed or improved and what other new functionality is needed. The development team of the infrastructure together with the end-user physicists get together to fix the work plan for the next release.
- Software weeks. Three or four times a year (coinciding with a release of the infrastructure software) we have a software week. Each subdetector reports on their progress on implementing their software using the infrastructure software and their immediate future plans.
- Sub-detector software reviews. These will happen a couple of times per year and the design of the new OO software of a subdetector will be reviewed in order to ensure that the concepts and design features are well understood and adopted. These reviews also permit to see if the framework is not adapted to the real needs of subdetectors.

2.8 What requirements do various software milestones, and other experiment milestones, place on functionality and timescale for delivery of core infrastructure components? Will these be met?

The experiment milestones, mainly subdetector’s Technical Design Reports (TDR’s), influence a great deal the requirements, priorities and timescales that are put on core infrastructure components. The coupling is due, in one hand, to the availability of people (physicists developers) for developing the new software using the provided infrastructure and in the other hand the need to produce results for the detector studies required by the TDR.

For some of the subdetector TDR’s (the ones that need to be submitted this year) it has been decided to use the old FORTRAN algorithms wrapped into the new GAUDI framework because of lack of time and man-power to re-implement them in C++ within this timescale. This requires that the infrastructure has been adapted to allow the use of existing FOTRAN algorithms with minimal code changes.

2.9 Which technical criteria will be used to decide when to use an existing commercial or HEP-built product and when to invest manpower in building an experiment (or HEP-wide) tailored solution for an infrastructure component, such as persistency? When will such choices be made?

The approach taken by LHCb of first defining an architecture allows us to provide implementations of the defined components which can vary over time and be adaptable to the more demanding requirements. For example, for the first releases of the framework, the persistency implementation has been based on the ROOT I/O system.The persistency implementation could be based on a OODBMS if required without having a big impact in the other parts of the system. Having designed this built-in flexibility allow us to delay the decisions on concrete technologies and implementations until a complete risk analysis and evaluation can be done.

LHCb being a small experiment in comparison with the two general purpose LHC experiments, is clear that the choice of the other experiments for adopting a given solution (commercial, HEP-made or other) plays an important role in our decision. The long term support for the chosen solution is one of the main factors in our decision. Therefore, if the solution is adopted by another large experiment is a quite strong point in order to guarantee such desired long term support.

3 Simulation and physics

3.1 What are the plans and status of GEANT4 migration and validation? What are the plans and responsibilities of the experiment in this validation process? How is the collaboration and software MOU process for GEANT4 working and what needs to be changed?

In LHCb we have not yet started to move to GEANT4. Our simulation program is still in Fortran based on Geant3. We intend to use GEANT4 within the GAUDI framework with a first version available by the end of the year 2000. We will validate the results with testbeam data and comparison with our Fortran simulation program.

There are 3 areas of activity concerning GEANT4 in LHCb at the moment. Our activity essentially started in the summer of 1999 so our experience is still very limited:

1. Together with Gunter Folger, Gonzalo Gracia has worked to port Geant4 to NT. There was also a learning exercise in the details of G4 and the testing procedure. Gonzalo also fulfilled the role of a naive user compiling and running systematically all the examples and tests and found several problems in the way this part is organised. Gonzalo is now a member of the “testing and quality assurance group”. In December Gonzalo worked on the release of G4 version 1.0. He made all the tests and ran all the examples and found several problems on NT. At that time he was the only person testing the release on NT. Immediately after the release he worked to make G4 compatible with the C++ ISO/ANSI standard. The only two compilers which were supporting the standard were on NT and on DEC (with a special compilation flag). Eventually a third compiler, the new compiler from Sun was also used for these tests. A significant number of small modifications of the code were made to complete the port.
2. Ivan Belyaev has made a small prototype to run G4 together with our software framework (GAUDI). This work was completed on Linux. Geant4 has been configured as a GAUDI “Service”. The integration allows options for configuring GEANT4 within GAUDI, to provide G4 with generator events and to retrieve events and hits from the G4 service. A number of simple examples have been made for the benefit of physicists starting to use this new simulation framework.
3. The calorimeter group have taken the work done by Vanya and have started to study the prototype of the electromagnetic calorimeter module presently under test in the testbeam. The aim is to compare the G4 results with those from the G3 simulation as well from the real testbeam measurements. This work is in progress and a status report is expected shortly.

How is the collaboration and software MOU process for GEANT4 working and what needs to be changed?

Since we have little experience it is probably premature to comment too much. However we can make a number of observations that give rise to some concern.

1. we notice that the uptake of GEANT4 in the collaborations seems to be rather slow. As far as we know no experiment uses GEANT4 in production yet.
2. although contributions from the experiments are encouraged, these contributions are not always perceived to be received with active interest by the GEANT4 development team. This tends to discourage further active participation.
3. the transparency of decision making is not always evident. The follow-up of decisions taken in the TSB are in the hands of the CORE developers, and do not always seem to get the priority placed on them by the TSB.

These worries can best be addressed if GEANT4 developers concentrate their efforts on helping the experiments launch their GEANT4 activities. This would give a good base of users and generate a lot of much needed feedback. It would ensure that further consolidation of GEANT4 is driven by the needs of real users.

Our comments concerning the MOU can best be explained by reference to Figure 8. This shows a standard schema for projects involving users who require a product and developers who devise a project to deliver what is needed. The main roles and the bodies for managing the activity are represented on the figure.

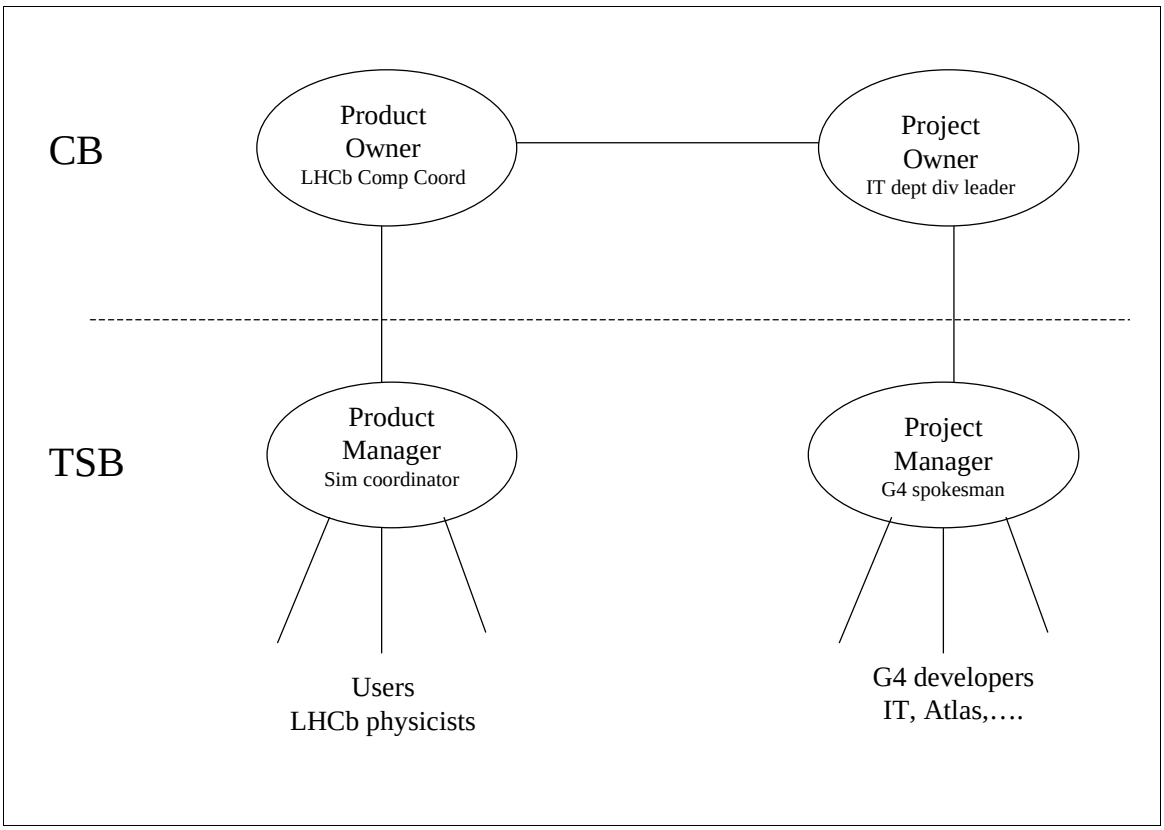

Figure 8 Project Management model for GEANT4

We would make the following remarks concerning the MOU:

- We believe that the MOU should be between institutes and labs that provide the manpower for development. Signatories to the MOU should be those who manage resources (managers from labs and institutes).
- The fact that some developers have an experiment label is irrelevant.
- A GEANT4 support service is needed in each major lab which hosts experiments that use it (CERN in our case).
- The development of GEANT4 should be use-case driven. Experiments have an obligation to define their requirements and the G4 collaboration to deliver the product. The link between users and developers needs to be strengthened. Each person should play a well-defined role (user or developer).

3.2 How is the experiment geometry specified? Is there one common specification in use for reconstruction, GEANT4 and other simulation?

How is the experiment geometry specified?

We have recently developed a specicialized framework [5] for entering the detector description. This framework covers all the information about detector apparatus including its logical structure, geometry, materials, conditions, electronics channels mappings etc. The users can access the geometry description through hierarchical description of the detector logical structure. Each detector module has its associated geometry information consisting of volume definition which includes the shape definition, material and list of daughter volumes. The geometry description is not a set of distinct islands of geometry data, but is organized as hierarchical structure. This hierarchical structure allows to traverse the geometry hierarchy and answer the various questions like location of an arbitrary point in the space inside geometry hierarchy, transformations between systems of coordinates etc. The user can define shape and dimensions of geometry volumes using a set of primitive solids (box, tube,...) or boolean solids (union, subtraction and intersection).

The current implementation for the persistent description of the detector data (e.g. the geometry data) is based on text files whose structure is described by XML (eXtendible Markup Language). XML is an application independent format for describing data. It has recently acquired wide support and is an industry standard issued by the W3 Organization.

Is there one common specification in use for reconstruction, GEANT4 and other simulation?

The answer is yes. The design of the detector description framework took this into account since the beginning. It was defined to be the single source of information about detector apparatus serving data to all of the data processing applications like simulation, reconstruction, analysis and DAQ, event display, etc. The designed model provides the information in the form which can be translated (via GAUDI

converters) to the formats required by external software frameworks like GEANT4, GEANT3. For example material and geometry classes are 100% compatible with those of GEANT4 while preserving enough data for their conversion into GEANT3 as well.

3.3 Upon which Generators is the experiment relying? Where and how will the long-term support for these generators come from? How will they be interfaced to the experiment's software suite and who is responsible for this?

Upon which Generators is the experiment relying?

Table 3shows our current plans for what event generators we are going to use for the next coming years.

Table 3 Expected usage of generator versions for LHCb

Year	1999	2000	2001	2005
Generator/Decay	P5/P5	P6/Q9	(P6,H5)/Q9	(P++,H++)/BPACK

P5 Pythia 5.7(24) - developed and maintained at Lund (Sjostrand)

P6 Pythia 6.1(34) - developed and maintained at Lund (Sjostrand)

P++ Pythia 7.x - under development at Lund (Lonnblad)

H5 Herwig 5.8 - developed and maintained at Milano/Cambridge (Marchesini, Webber)

H++ Herwig++ - under development at RAL/Cambridge (Seymour, Webber)

Q9 QQ 9.2 - developed by CLEO Collaboration (David Jaffe), co-maintained by CDF Collaboration (Lyn Garren)

BPACK EvtGen >1.0 (A.Ryd) or new package eventually based on EvtGen and developed by CMS, ATLAS and LHCb. This is being discussed right now: http://home.cern.ch/m/msmizans/www/production/Bpack

Where and how will the long-term support for these generators come from?

Pythia: Lund (Sjostrand, Lonnblad)

Herwig: Milano/RAL/Cambridge (Marchesini, Seymour, Webber)

QQ: Cleo collaboration, CDF collaboration, LHCb collaboration (hopefully from the year 2002 QQ will be needed just for cross checks and a frozen version will be used).

BPACK: BaBar or CMS+ATLAS+LHCb (eventually + BABAR)

How will they be interfaced to the experiment's software?

Interfacing the generators to Fortran is done using StdHep 4.06 (Lyn Garren). The C++ interface will be done using StdHep >4.07 (Lyn Garren) or HEPMC >0.9 (Dobbs, Hansen) or whatever else fit to our exigences.

Who is responsible for this?

The GAUDI + generator groups should share the responsibility, tough for now there's no dedicated manpower in the LHCb generator group. General problems related to the generator interfaces are faced in BPACK.

General comment: C++ generators and interfaces are in late with respect to other part of the simulations (i.e. the first - probably bugged - versions of them are not yet available). It is difficult to take any decision at this time (which interface, which generator model, which decay model to adopt).

3.4 What are the experiment's plans for fast simulation?

We have no plans for fast simulation. The question has not been raised yet in LHCb. With the experience we have from a LEP experiment (ALEPH) we can suggest to LHCb to have only one simulation program: with a detailed simulation where it is necessary and a parametrization of the response every time it is possible.

3.5 How does the interface with physics groups work? How is the responsibility divided between Software and Computing groups and "Physics groups" for physics algorithms, physics object definition and identification?

The LHCb collaboration is in the process of writing Technical Design Reports for the different detectors and most of the physics studies done are tailored to the optimization of the detectors. As a consequence there are no formal "Physics Groups" but people performing physics studies belong to Detector groups. Physicists working on software have frequent interactions with the Software and Computing group both via formal meetings and personal interactions.

Since LHCb is a relatively small collaboration it is possible for members of the Detector groups to have a personal interaction with specialists in the computing groups about the technical aspects of the software they need to develop. There is no formal division of responsibility between the Detector Groups and the Computing group. In practice the Computing group provides the frameworks and infrastructure and the support for software development, specifying some basic structure the algorithms and the data have to conform to. The Detector groups develop the actual physics content of the software, building on the basic structure, and provide feed-back, based on their physics need, about the structure itself allowing a cyclic evolution of the software.

3.6 What Mock Data Challenge, or other activities, which exercise the full spectrum of software from simulation through to physics object data, is being planned? How will the success of these exercises be assessed?

3.7 What is the decomposition into trigger, reconstruction, simulation and physics analysis software? What are common parts? How do you expect that the current choices will evolve? Which decisions do you foresee in the future?

The trigger, reconstruction, simulation and physics analysis software are considered to belong to different processing stages. In each stage data that is “consumed” by at least one of the other stages is produced. Simulation and trigger produce data that constitute the input to the reconstruction that in its turn produces the input data for physics analysis.

The common parts of the of the software for the different processing stages are the foundation libraries and the various frameworks (in particular, the main framework, GAUDI, for the new software) they are built on. Figure 4 shows the structure of the software and how the different data processing applications are build on top of foundation libraries and frameworks. The idea is that all the event data processing applications (trigger, simulation, reconstruction, analysis) are build on top of the same software components.

The current Foundation libraries are STL, CLHEP and NAG. The frameworks ensure the commonality of interfaces and of use of basic services. Packages providing the functionality for a specific service are “plugged” in an application through the framework. This “plug-and-play” mechanism allows different applications to use only the packages that provide the necessary services. In addition it allows to employ, for a specific service, a package different from the default. As a consequence the concrete packages providing the services can be common or not depending on the needs. The idea is to maximize the common packages. In particular, we would like to use the same data management packages, data persistency packages, user interface packages, etc. in all applications.

Some of the external packages we have currently chosen are HTL for histogramming, GEMINI for fitting and minimization, XML for detector description.

We expect that the choice of the concrete packages for the different services will be re-evaluated and could be changed when new packages become available (e.g. visualization).

In the future we foresee to choose a default implementation for the services were the choice is still open. This could be different for the various processing stages if the needs are shown to be different (e.g. Data Persistency for raw data and physics analysis data).

4 Non-experiment specific software

4.1 Upon which commercial, public domain, or HEP community software is the experiment relying?

The current (FORTRAN) software relies heavily on CERNLIB and ZEBRA. It is expected that we will continue to generate and analyse GEANT3/ZEBRA based simulated data until at least the end of 2001.

In future, we plan to use in our software any third party components that satisfy our requirements. Our software architecture has been designed to shield our applications from the specific choice of components, via the use of abstract interfaces. This choice makes it relatively easy to integrate third party components, and to migrate to alternative components if better implementations become available or if products become obsolete or unmaintainable.

Currently we are either using or planning to use the following third party components:

- Event generators (see LHCb internal note 99-027, and see also Section 3.3). We currently use PYTHIA 6 for event generation, and the CLEO Monte Carlo QQ for decays. These are interfaced to our GEANT 3 based simulation program via the STDHEP interfaces. We plan to benefit from the developments of these generators, in particular concerning future C++ implementations.
- Simulation. Our current simulation program is based on GEANT 3. We are members of the GEANT 4 collaboration and plan to migrate to GEANT 4 during 2001 (see also Section 3.1)
- Foundation libraries. We plan to use foundation libraries that have wide acceptance within the HEP community. Currently we rely on the C++ standard template library, CLHEP, NAG C. We intend to access other third party components through the AIDA interfaces which are currently being defined in the context of LHC++.
- Data storage. We have designed our software so as to shield our applications from the storage technology adopted for our data. This flexibility will allow us to use the same persistency scheme adopted by other LHC experiments. Currently we use both ZEBRA and ROOT to store our event data and Oracle to store bookkeeping data. Geometry data is stored in XML format, which we parse using a open source XML parser. See Section 3.2.
- Data distribution. We assume that tools will be provided for distributed data access, remote job submission etc. - we expect this to be an outcome of projects such as MONARC and the Grid initiatives, in which we are participating.
- We have not yet chosen a product for handling interactive analysis (GUI and visualisation), but we intend to use public domain and/or HEP software for this. Currently we use both PAW and ROOT for histogram and Ntuple analysis and storage.
- We use CVS for code repository, and CMT for configuration management. Our development environment is Visual C++ on WNT, and GNU tools on RedHat Linux. We rely on AFS to provide a unified file system across the two platforms. We use Framemaker for documentation, and Object Outline for code documentation. Our design tools are Visual Thought and Rational Rose. We plan to use the forthcoming CERN-IT Remedy service for bug reporting and tracking.

- We use Microsoft tools for our web server (e.g. FrontPage) and ColdFusion coupled with MS Access for our collaboration database, though we plan to migrate soon to the PIE application provided by CERN AS Division.

4.2 What service and support agreements are either in place, or need to be put in place? How will this be done? What is the role of CERN IT in this?

It is essential that software used in the experiment should be supported on the chosen operating systems, with new releases closely following the introduction of new operating system/compiler versions. It is also very important to have access to the source code, either directly or through escrow agreements that would give access to the source code should the provider decide to no longer support the product, or to not support a necessary operating system/compiler version within an acceptable delay.

It is our intention to use, as far as possible, components that are supported or recommended by CERN-IT or by the wider HEP community. We expect CERN-IT to distribute CERN supported/recommended software in compiled form for the supported platforms/compilers, and to negotiate the necessary licensing and support agreements. Access to this software must be on equal terms for all collaborating institutes. In addition CERN IT should provide the infrastructure for the distribution of software which is of interest to the wider HEP community but which is contributed and/or maintained by entities other than CERN-IT, even if not officially supported or recommended by CERN-IT. The role of CERN-IT in such a case is to act as a central distribution point, but with no commitment as to the content or support of the software.

Support for third party software which is needed only by LHCb should be arranged within the collaboration (e.g. customisation of the CLEO QQ Monte Carlo is currently ensured by the LHCb Lausanne group).

4.3 What risk factors does the experiment face in its use of such commercial, public domain or HEP community software? What plans does it have to mitigate the risks?

For commercial software, we see the following risks:

- The company ceases to support the software (either through choice, or because it ceases to exist)
- The company decides not to support the product on the required platforms/compilers, or does not release a version for these platforms in a timely fashion, or other commercial products in use require incompatible platform/ compiler version.
- The software misses some required functionality, which the company does not agree to add in a reasonable time-scale or at a reasonable price. Similarly the company may not fix bugs in a reasonable time-scale

The above points should be addressed when negotiating purchasing and licensing agreements. We rely on CERN-IT to conduct these negotiations for the software it recommends, particularly where niche products are involved. For mass market products, the possibility always exists to switch to an alternative product - our software architecture is designed to make such switching relatively painless.

The same arguments apply to public domain and HEP community software, where one has no leverage on the authors to provide the required support. In this case we must make sure that the source code is publicly available, preferably following the Open Source model, so that we may if necessary apply our own changes, and feed them back to the common code base. The risk here is that we end up maintaining a large part of this software - this can be mitigated by choosing to use software which is accepted by as wide a community as possible and, for what concerns HEP-code, to form collaborations to maintain and develop this software. CERN-IT should play a major role in fostering such collaborative efforts.

One particular worry is in the area of mass storage. Although our software shields us from the underlying technology, any change in technology implies the migration of large quantities of data. Therefore we intend to use the same technology adopted by the other LHC experiments, so that only one migration solution would have to be found. We expect CERN-IT to play a major role in the initial risk assessment exercise and in organising any future migration.

4.4 What experiment-specific R&D needs to be undertaken in order to make either design or implementation choices for the software? How is this proceeding and when are the decision points?

The GAUDI architecture and framework are a prototype to be used for implementation of the first OO version of the experiment software. The framework is in place and OO reconstruction algorithms are being developed within this framework. We plan to have a complete OO software analysis chain towards the end of 2001, when we can review the situation and begin work on a production implementation. We are at the same time investigating the suitability of JAVA for at least part of these applications.

We hope to benefit from developments throughout the HEP community while at the same time contributing to these developments where we feel we can make a useful contribution.

4.5 Is the experiment relying on technology, or functionality, which is assumed, will be provided by others and which currently is still in the R&D stage --- e.g. grid services? If so, what are the risk factors?

We cannot give a reasonable answer to this question at this stage. Nearly everything to be used in 5 years time is in the R&D stage, and nearly everything other than GAUDI is provided by third parties!

5 References

1 GAUDI: Scenarios and Requirements, LHCb 98-065 COMP, (http://lhcb.cern.ch/computing/offline/pdf/gaudiscenarios.pdf)

2 Object Solutions

3 Ivar Jacobson, Grady Booch, James Rumbaugh, *The Unified Software Development Process*, Addison Wesley, 1999

4 F. Ranjard, et al., *Use of a Software Configuration Management Tool in LHCb*, proceedings of CHEP2000, Padova, (see also http://chep2000.pd.infn.it/paper/pap-f151.pdf)

5 R. Chytracek, et al., *The LHCb Detector Description Framework*, proceedings of CHEP2000, Padova, (see also http://chep2000.pd.infn.it/paper/pap-a155.pdf)

A GAUDI Project Plan and Joblist

Figure 9 Fragment of the Gantt chart for the GAUDI project

B Major Milestones to TDRs

Magnet

Freeze design Oct 1999

TDR tender out Dec 1999

Vertex

Design of silicon det Jun 2000

TDR Apr 2001

ITR

freeze design Jun 2001

TDR Sep 2001

OTR

TDR Mar 2001

RICH

complete design Mar 2000

TDR Jun 2000

Muon

choose technologies Jan 2000

TDR Jan 2001

Calorimeter

Eng design Apr 2000

TDR Jul 2000

Trigger

L0/L1 TDR Jan 2002

DAQ

TDR Jan 2002

Computing

Finish first prototypes Jul 2000

TDR Jul 2002