Propriété littéraire et scientifique réservée pour tous les pays du monde. Ce document ne peut être reproduit ou traduit en tout ou en partie sans l’autorisation écrite du Directeur général du CERN, titulaire du droit d’auteur. Dans les cas appropriés, et s’il s’agit d’utiliser le document à des fins non commerciales, cette autorisation sera volontiers accordée.

Le CERN ne revendique pas la propriété des inventions brevetables et dessins ou modèles susceptibles de dépôt qui pourraient être décrits dans le présent document; ceux-ci peuvent être librement utilisés par les instituts de recherche, les industriels et autres intéressés.

Cependant, le CERN se réserve le droit de s’opposer à toute revendication qu’un usager pourrait faire de la propriété scientifique ou industrielle de toute invention et tout dessin ou modèle décrits dans le présent document.

© Copyright CERN, Genève, 1998

ISSN 0304-2898
ISBN 92-9083-136-7

Literary and scientific copyrights reserved in all countries of the world. This report, or any part of it, may not be reprinted or translated without written permission of the copyright holder, the Director-General of CERN.

However, permission will be freely granted for appropriate non-commercial use.

If any patentable invention or registrable design is described in the report, CERN makes no claim to property rights in it but offers it for the free use of research institutions, manufacturers and others. CERN, however, may oppose any attempt by a user to claim any proprietary or patent rights in such inventions or designs as may be described in the present document.
1998 CERN SCHOOL OF COMPUTING

Funchal, Madeira, Portugal
6–19 September 1998

PROCEEDINGS
Editor: C.E. Vandoni
Abstract

The programme of the School was arranged around four themes. In addition there was an evening lecture on quantum computing.

The theme “Intelligent Monitoring and Control” covered the applications of artificial intelligence techniques to the data-acquisition, monitoring and control of large experimental systems in high-energy physics. The complexity and expected long lifetime of the next generation of HEP experiments, in particular those being developed for the Large Hadron Collider (LHC) at CERN, present a major challenge. Diagnosis, alarm handling and user-assistance tools, all extracting information from a common knowledge base, can help to solve problems when the human experts are unavailable.

A number of HEP experiments due to start in about the year 2000, and those planned for the LHC later, will extend considerably the already high data volumes. The LHC experiments are expected to accumulate some 5 PB ($10^{15}$ bytes) of data per year, giving rise to a total of perhaps 100 PB over their 15–20 year lifetime. Apart from the sheer volume of data, the extremely long time-scales compared to the standards of the IT industry or to the career of an individual physicist, require the use of standard and, if possible, commodity solutions. The “Petabyte Storage” theme covered the basic components that are being investigated for the storage and management of data. These include the High Performance Storage System (HPSS), a mass-storage system built according to the IEEE Computer Society reference model, and Objectivity/DB, an ODMG-compliant Object Database Management System (ODBMS). The RD45 project, which is investigating these components as a solution to LHC event data handling, was described.

The theme “Agent and Distributed Computing Technology” explored the field of distributed computing with emphasis on high performance and agent technologies. The problems to be resolved were defined and the underlying technologies described, including the use of toolkits such as DCE/CORBA. The use of agents written in Java for distributed physics analysis was examined, including the use of submission and compute servers and algorithm agents.

Starting from the standard software process model (CMM) of the Software Engineering Institute, the theme “Software Evolution” explained the activities required to improve the process, and enlarged on configuration management and quality through software metrics. For the exercises, students worked on their own projects, applying software metrics to the source code and organizing all the development documents with a version control system (CVS).
Preface

The 21st CERN School of Computing took place at the Hotel Savoy, Funchal, Madeira, from 6-19 September 1998. It was organised by CERN in collaboration with LIP, Lisbon and the University of Madeira, Funchal.

13 lecturers, of which seven were from CERN, four from the USA, one from Portugal and one from the UK, were invited to give courses at the School. 67 students, 14 of whom were funded by the European Commission and by UNESCO, attended the School. The students came from 45 institutes from 22 countries and were of 22 nationalities.

The programme of the Schools was organised round four themes:

- Agent and Distributed Computing Technology
- Intelligent Monitoring and Control
- Petabyte Storage (data bases)
- Software evolution

Each theme was structured so that the lectures in the first week were more general in their coverage with the second week’s lectures more specialised towards computing problems relevant to particle physics and the LHC programme. It should be noted that practical exercises have become an important part of the scientific programme and require a complex computing infrastructure.

The School was opened in the presence of Prof. Rui Carita, University of Madeira, Dr. Raimundo Quintal, Municipality of Funchal, Dr. Conceição Estudante, Regional Direction of Tourism, Dr. Carlo Vandoni, CERN, Prof. Gaspar Barreira, LIP and Prof. João Varela, LIP and CERN.

The following members of the Advisory and Local Organising Committees attended the School for a few days at various times: G. Barreira (Local Organising Committee), F. Etienne (Technical Adviser), F. Flückiger L.O. Hertzberger, G. Kellner (who was also a lecturer) and J. Varela (Chairman of the Local Organising Committee). C.E. Vandoni was the Director of the School.

The computing and peripheral equipment was provided by and via CERN. Equipment lent by various computer manufacturers was delivered to CERN where it was set-up, tested, then dismantled and sent, by lorry and by boat, to Funchal. Software and licences needed by the lecturers for their presentations were kindly provided by various software companies and our thanks go to them for their help. Our Portuguese colleagues did a wonderful job in the provision of the necessary network connection from Funchal via the University of Madeira, Lisbon and CERN and, together with some students from the University of Madeira, gave very efficient help in the unloading and installation. The setting up of a complete computing facility was rewarded by the high satisfaction of the students and of the lecturers. This setting up was not easy, and needed the collaboration of many people.
Fabien Collin of the CERN, IT-PDP group, and Andreu Pacheco of the University of Barcelona were the System Managers of the CSC computer centre and their very hard work and efficient management, both before, during and after the School was a major contributing factor in the success of the School. J. Gomes, J.-C. Nogueiro and J. Conceição (LIP, Lisbon) contributed greatly, together with our two Systems Managers, in the setting-up and installation of what became known as "Cap Canaveral". This name was coined by our Savoy hotel contact, Mr. Antonio Correia.

Dr. J. Gajewski (University of Warsaw), Chairman of the Local Organising Committee for the 1999 School, attended during the first week, so as to obtain a better idea of the organisation and setting up of a CERN Computing School.

Mrs. P. Loureiro gave organisational and secretarial assistance and her efficiency, hard work and friendly character was greatly appreciated by all.

Very special thanks must go to the lecturers for the enormous task of preparing, presenting and writing up their courses.

Finally, the participation of so many people coming from many different parts of the world, is a convincing proof of the usefulness and success of this School.

We express our gratitude to our secretary and administrator, Jacqueline Turner, not only for the efforts made during the preparation of the School, but also for her invaluable help in preparing these Proceedings.

Finally, we should also thank the students for their enthusiasm and active participation and we wish all of them success in their professional life.
Advisory Committee

W. Carena, CERN, Geneva, Switzerland
F. Etienne, Centre de Physique des Particules de Marseille, Marseille, France
F. Flückiger, CERN, Geneva, Switzerland
L.O. Hertzberger, Amsterdam University, Amsterdam, The Netherlands
A.J.G. Hey, University of Southampton, Southampton, U. K.
G. Kellner, CERN, Geneva, Switzerland
J. May, CERN, Geneva, Switzerland
P. Palazzi, CERN, Geneva, Switzerland
J. Shiers, CERN, Geneva, Switzerland
J. Turner, CERN, Geneva, Switzerland
C.E. Vandoni, CERN, Geneva, Switzerland
J. Varela, CERN, Geneva, Switzerland and LIP, Lisbon, Portugal
D.O. Williams, CERN, Geneva, Switzerland
Lecturers

K. Ackerstaff, CERN, Geneva, Switzerland
A. Aimar, CERN, Geneva, Switzerland
A. Cardoso, University of Coimbra, Coimbra, Portugal
M. Dönszelmann, CERN, Geneva, Switzerland
D. Düllmann, CERN, Geneva, Switzerland
L. Guzenda, Objectivity Inc., Mountain View, USA
A. Hey, University of Southampton, Southampton, U. K.
R.G. Jacobsen, University of California, Berkeley, USA
G. Kellner, CERN, Geneva, Switzerland
C. Kesselman, University of Southern California, Marina del Rey, USA
A. Khodabandeh, CERN, Geneva, Switzerland
P. Mato, CERN, Geneva, Switzerland
R.W. Watson, Lawrence Livermore National Laboratory, Livermore, USA
Contents

Abstract ........................................................................................................ iii
Photograph ....................................................................................................... iv
Preface ........................................................................................................... v
Advisory Committee ....................................................................................... vii
Lecturers ......................................................................................................... viii

INTELLIGENT MONITORING AND CONTROL

Intelligent Monitoring and Control
   A. Cardoso ................................................................................................... 1

Application of Expert Systems in High Energy Physics: the ALEPH and ZEUS Case
   P. Mato ...................................................................................................... 29

PETABYTE STORAGE

The Architecture of the High Performance Storage System (HPSS)
   D. Teaff, R.W. Watson, B. Coyne ......................................................... 41

Building Petabyte Databases with Objectivity/DB
   L. Guzenda ................................................................................................. 65

Object Databases as Data Stores for High Energy Physics
   D. Düllmann .............................................................................................. 71

AGENT AND DISTRIBUTED COMPUTING TECHNOLOGY

Computational Grids
   I. Foster, C. Kesselman ............................................................................... 87

Agents - Mobile Agents in Java
   M. Dönszelmann ....................................................................................... 115

SOFTWARE EVOLUTION

Software Evolution - Track Introduction
   A. Khodabandeh ...................................................................................... 119

User Requirements
   G. Kellner .................................................................................................. 123
Project Management and Tracking, Resource Estimate

A. Aimar .................................................................................................................. 127

Introduction to Software Documentation

A. Aimar .................................................................................................................. 135

Quality Through Software Metrics

A. Khodabande ........................................... ......................................................... 143

Configuration Management

R.G. Jacobsen, D. Johnson ...................................................................................... 147

Software Evolution: Case Study, OPAL

K. Ackerstaff ......................................................................................................... 159

EVENING LECTURE

Quantum Computing: An Introduction

A.J.G. Hey .............................................................................................................. 165

List of Students .................................................................................................... 181
INTELLIGENT MONITORING AND CONTROL

Amílcar Cardoso
AI Lab, CISUC - Center for Informatics and Systems, University of Coimbra, Portugal

Abstract
Human operators of monitoring and control systems typically perform knowledge intensive tasks requiring human intervention in diverse and sometimes critical situations. The cognitive overload to which they are submitted may motivate human errors which compromise safety and performance. Expert Systems offer the means to codify and automate the use of the knowledge of the best experts in a given domain, and may provide support to monitoring, diagnosis and control tasks. In this text we will make an overview of what expert systems are, what they may offer and what they don’t. We will present their typical architecture and the most common knowledge representation formalisms and reasoning mechanisms, and discuss how they may deal with uncertainty. The life-cycle of an Expert System will also be presented, special attention being devoted to knowledge acquisition activities. An overview of some applications of expert systems to monitoring and control will be given, as well as some alternative approaches.

1. INTRODUCTION

The control needs of complex and expensive processes have motivated the development of sophisticated computational systems which allow the monitoring and control of complex processes by small crews of specialised human operators. Typically, these operators follow the process evolution through a set of instruments connected to a computer system which interacts with sensors, alarms and actuators. The control system is intended to minimize human intervention and to provide the operators with the maximum information on the controlled process. The operators’ task essentially consists in trying to minimize failure risks while maximizing performance. The required degree of expertise depends on the complexity of the process and of the control system, on the importance, to people and equipment safety, of keeping the process in normal operation, and on the losses that an incorrect process control or its interruption may cause.

In normal steady-state conditions, the control system takes charge of keeping the system in a given set point, and the operators confine themselves to monitoring the most important variables. Human intervention is required during system start-up and shut-down, and also to regulate the process to given resources and needs.

Another, more critical, kind of situation occurs when equipment or process failures, or human errors, originate anomalous conditions. In such circumstances the control system presents a great quantity of information to the operators in a short time, originating in the most diverse sources, with the process conditions quickly changing and alarms escalating. In such a stressed situation, operators must discriminate actual faults from false alarm states, perform malfunction diagnosis, and plan and accomplish corrective actions. This requires them to intensively resort to their knowledge on the process and on the control system. In particularly complex, stressed or time-critical situations, they

---

1 According to [1], “monitoring activities track process variables intelligently and provide knowledge-based explanations of normal process behavior. (...) Fault detection (...) involves differentiating between normal and abnormal conditions. (...) Malfunction diagnosis isolates and identifies process malfunctions. (...) Corrective-action planning takes a diagnostic conclusion and provides a plan of action to deal with the problem safely and economically.”
have difficulty in recognising the most relevant data, to evaluate the process state and to identify causes of the troubles. Consequently, they tend to perform inaccurate corrective actions and answer the problems in a wrong sequence. The cognitive overload they are submitted to withholds them to execute a correct, consistent and quick course of actions (see [2], [3] and [4] for actual examples).

Moreover, the more complex the process is, the most difficult becomes to the operators to understand its dynamic behaviour, particularly in unforeseen situations, and the most difficult becomes to guarantee a uniform level of expertise between different operators.

All these factors account for the need to endow control systems with tools aimed at giving online support to the operators’ decision process, reducing the demanded cognitive load when abnormal conditions occur [2, 5].

Conventional programming techniques may be used to build such tools. However, they require quantitative models of the controlled processes, describing their whole range of operation, which usually is, in practice, an impossible task. Even when such models exist, they often demand the access to hard to obtain parameters [6]. Moreover, the resulting programs exhibit forbiddingly higher execution times with the increase of the process complexity [3].

Artificial Intelligence presents a set of favourable answers to these problems. Expert systems, in particular,

i) offer the means to codify the knowledge of the most expert operators and process designers, and

ii) to computationally automate its use, making it permanently available.

Their symbolic processing capability makes them a viable alternative to traditional monitoring and diagnosis methods, particularly when the controlled process is complex, difficult to model in a quantitative way, and the available knowledge is incomplete.

This paper intends to give an overview of how expert systems may contribute to improve monitoring, control and diagnosis. In the following section we will give an introductory picture of what expert systems are, what kind of applications they are intended to, and what kind of obstacles to their development are to be considered. In Section 3 the typical architecture of an expert system will be presented, as well as the main approaches to knowledge representation and reasoning. In Section 4 we will present some clues on how expert systems may deal with uncertainty. Section 5 will be devoted to the development of expert systems. In Section 6 we will return to monitoring and control, to analyse drawbacks of the expert systems approach, to point some emerging alternatives and to give an overview of the main applications of expert systems to those kinds of tasks. At last, in Section 7 we will draw some conclusions.

2. EXPERT SYSTEMS APPROACH

2.1 Characterisation of Expert Systems

Expert systems are knowledge-based computer programs, in the sense that they explicitly incorporate domain knowledge. They are intended to exhibit problem solving capabilities comparable to those of human experts. To attain a high level of expertise, they are restricted to solving problems in narrow knowledge domains, in contrast with some other knowledge-based programs which adopt a generalist approach. The knowledge they embody is acquired from human experts, and specific methodologies exist to guide this knowledge acquisition process. Once knowledge is incorporated, the program applies symbolic reasoning algorithms to infer new information from available knowledge and data.

Expert systems have the ability

i) to incorporate expert knowledge, including heuristics and rules-of-thumb,

ii) to deal with uncertain information and incomplete knowledge, and
iii) to explain the reasoning process underlying a founded solution.

They also offer easy integration of new knowledge and maintenance of the existing one. These characteristics make them especially useful when
i) there is an insufficient number of experts or they tend to disappear
ii) experts are needed in different places or in hostile environments
iii) experts are subjected to cognitive overload

2.2 Applications

Typically, applications of expert systems fall into two classes of problems: classification and construction.

In classification problems, a set of data is given and the expert system attempts to find which of a predetermined number of classes the data corresponds to. Medical diagnosis, equipment diagnosis, plant classification and process monitoring are examples of classification problems.

In construction problems, the expert system is given a goal and a set of constituent parts, and attempts to build an arrangement of the parts which may attain the goal. Examples are product design, computer configuration, route planning and production scheduling.


Depending on the concrete problem to solve and on the environment where an expert system is expected to operate, we may consider several possible configurations for it: the expert system may work in a stand-alone fashion, interacting only with human users; it may be integrated in a computational environment, interacting with it and also with human users; it may be completely embedded in the computational environment.

2.3 Applications in Monitoring and Control

Expert systems for monitoring and control applications must satisfy some specific requirements [13, 14]:

i) they must dispose of efficient means to data acquisition;
ii) they must have the capability to deal with time varying data;
iii) the reliability of their answers must degrade in a graceful way when the available time to get the answers decreases;
iv) they must exhibit a robust behaviour: they must operate continuously while the process is running, without significant performance degradation when abnormal or unexpected conditions occur;

Although these requirements usually are not easy to attain, things get harder to manage when hard real-time constraints are present. Typically, in real-time applications many decisions are required in a timely fashion, facts have limited life spans, the process generates large amounts of data, operations are tied to clock time or time intervals, control decisions depend on current and historic data, and many asynchronous, concurrent operations may occur [15]. This makes the application of expert systems to real-time applications a current hot topic of research, despite the existence of several successful reported systems. We will come back to this topic later in Section 6.
2.4 The Knowledge Acquisition Bottleneck

One of the main obstacles to expert systems development lies in the knowledge acquisition task, aimed at eliciting knowledge from a human expert, and incorporating that knowledge in the expert system. This is a very hard and time consuming task, due to a series of reasons:

i) when the development starts, the expert system developer usually doesn’t know anything at all of the expert’s domain of knowledge, and must learn enough about the expert’s problem solving task to build the system; this is basically done through a series of interviews with the expert, which must be carefully planned to ensure successfulness;

ii) usually experts have great difficulties to understand what an expert system really is;

iii) in consequence, when describing to the expert system developer how they perform their activity, experts risk to emphasise irrelevant information and to neglect important one;

iv) sometimes, experts are not motivated to collaborate with the developers, or they are too busy to devote enough time to the interviews.

Given the importance of knowledge acquisition to the success of an expert system project, researchers and field developers have proposed several methodologies to guide not only this activity, but also the development process itself, as both are tightly linked. Considering the introductory character of this course of lectures, in this paper we will devote Section 5 to this topic, adopting pragmatic guidelines originating in field developers. We must stress, however, that more formal methodologies have been proposed, the most important of all is KADS [16, 17], which results from an ESPRIT funded joint effort.

3. EXPERT SYSTEMS ARCHITECTURE\(^2\)

An expert system is a program whose degree of expertise is intended to be comparable to that of a human expert in a specific domain of knowledge. To attain this purpose, they rely on the incorporation of expert knowledge which the program uses for problem solving.

Expert knowledge has some specificities which are worth mentioning:

i) it is seldom based on clear definitions or on precise algorithms;

ii) it is composed of general theories, but also of heuristics, which are used to simplify problem solving, to identify common situations, and so on;

iii) it is continuously changing.

These specificities influenced the architecture of expert systems. Actually, a uniqueness of expert systems lies in the clear separation between knowledge, on one hand, and algorithms to apply knowledge, on the other hand. This separation results in the existence of two main modules (see Figure 1): the Knowledge Base (KB) and the Inference Engine (IE). To complete the basic architecture, a User Interface (UI) guarantees communication with the outside world.

This architecture greatly favours expert systems development and maintenance, because we may freely change the KB contents (the evolving part of the system) without having to re-program the remainder of the system. Also, we may develop efficient algorithms when building the IE, without caring with the specificities of a particular domain.

---

\(^2\) This section is based on [18] and [19].
The algorithms of the IE are in principle domain independent. Also, the UI may usually be composed by generic mechanisms. The KB is obviously domain dependent and actually determines the characteristic behaviour of a specific expert system. Roughly, given an expert system which solves problems in a domain A using the Knowledge Base KBA, if we substitute KBA with a Knowledge Base KBB describing domain B we will get an expert system which solves problems in domain B. This is the principle behind the idea of joining the IE/UI modules in one domain independent tool, called Shell. There are many commercially available Shells which may be used to build expert systems. With such a tool, the expert system builder may concentrate on the KB construction, and use the generic inference and interface mechanisms offered by the Shell, instead of programming the system from scratch.

3.1 Knowledge Base and Representation Formalisms

A Knowledge Base contains domain knowledge, which may be represented through different formalisms. Knowledge representation formalisms must simultaneously be clear and readable for the ES developer, be computationally workable, exhibit adequate expressive power, and be easy to update and edit. We are going to present some of the most common alternatives.

3.1.1 Object-Attribute-Value Triplets

One of the simplest formalisms is the OAV Triplet (see Figure 2), which may conveniently represent facts in the Knowledge Base. Given a relevant object of the domain (e.g., an aeroplane), we may define a set of attributes which characterises the object (e.g., the number of engines, the engine type and the wing design). Each attribute may have one or more values (e.g., number of engines = 4). Figure 3 illustrates how we may use OAV triplets to represent three facts about a specific aeroplane.

A common simplification of this formalism is the Attribute-Value Pair, which makes the object implicit. When using this representation, it is advisable to include some reference to the implicit object in the attribute name (e.g., “Number of Engines of C130 = 4”). Often this inclusion is mandatory because attributes in AV pairs must have unique names (this is not the case in OAV triplets, where different objects may have attributes with the same name).
3.1.2 Semantic Networks

We may consider Semantic Networks as a generalisation of OAV Triplets. A Semantic Net is a directed graph with labelled edges and nodes (see Figure 4). While an OAV triplet represents the relation between one object, one attribute and one value, a semantic net may represent several objects, several attributes per object, and even relations between objects. Relationships are of prime importance to organise and use knowledge. With the ability to relate objects, we may increase the expressiveness and versatility of the knowledge base. For instance, we may create nodes representing classes of objects sharing common characteristics, and link the nodes with sub-class/super-class relationships.
The semantic net of Figure 5 illustrates the use of two very useful relations: IS-A, which relates a class and a specific instance of that class, and A-KIND-OF, used to express sub-class relationships. These relations are used to represent inheritance between objects, which may significantly enhance expressiveness and reduce memory storage requirements.

3.1.3 Frames

Frames have similarities with semantic nets in that they offer the means to represent objects, their attributes and values, and relationships between them. Roughly speaking, they are sophisticated data structures composed of a set of non-homogeneous slots which may be filled with information, each slot corresponding to an object attribute. In contrast to the simple object representation that we have seen up to now, instead of storing just values, slots may contain relations, default values, pointers, rules and even procedures (see Figure 6). Frames may also be used to build semantic networks, offering a very rich representation for their nodes. Actually, we may think of a frame as an extension to the O-A-V representation.

![Figure 6 - A Frame](image)

Figure 7 illustrates the use of frames to represent the concept of *dog*. Relations, default values, ranges of values and procedures may be observed in this frame.

Frames are a powerful, robust, versatile and expressive knowledge representation formalism, but they are difficult to use and their behaviour is hard to predict. However, they have proved effective for large-scale, complex systems, involving default values and a large amount of a priori facts.

![Figure 7 - Example of a Frame](image)
3.1.4  **Predicate Logic**

Predicate logic is an extension of propositional logic which may be used to represent the components of a sentence. Its simplest form, the first order predicate logic, may represent rather complex sentences. For instance (example from [18]), we may write

- `mammal(dog)`, meaning “a dog is a mammal”, which is a true sentence;
- `four_legs(dog)`, meaning “a dog has four legs”, which is usually true;
- `mammal(chicken)`, meaning “a chicken is a mammal”, which is false;
- `sister(joan, jack)`, meaning “Joan is Jack’s sister”; we don’t know if this is true or not because it depends on who actually are Joan and Jack;
- `sister_of(steve, mary) AND cousin(mary, ann)`, meaning “Mary is Steve’s sister and Ann’s cousin”;
- `ill-humoured(boss) AND ask_for(boss, rise) ⇒ bad(idea)`, meaning “Asking for a rise when the boss is ill-humoured, is a bad idea”;
- `human(X) ⇒ mortal(X)`, meaning “All humans are mortal”

Predicate logic has the advantage of offering a well established syntax and a clear semantics for knowledge representation, and a founded method, called resolution, for inferring new information. However, implementing resolution is not a trivial task, so the use of predicate logic in commercial shells is not common. Even so, there are programming languages (e.g., Prolog [20]) which implement predicate logic allowing the development of expert systems using this knowledge representation formalism.

3.1.5  **Production Rules**

Production rules, usually called just Rules for the sake of simplicity, are the most used knowledge representation formalism in Expert Systems. They origin from theories of cognitive psychology which proposed production rules as models for knowledge storage in *long-term memory*. According to those theories, sensory input provides stimuli to the brain, triggering rules of long-term memory, resulting in the production of responses [21]. Rules are intended to model small chunks of knowledge, providing modularity to knowledge representation.

The most general form of production rules is:

\[
\text{Name: IF } \text{Antecedent THEN ConsequentA ELSE ConsequentB}
\]  \(\text{(1)}\)

where *Name* is a rule identifier, *Antecedent* has the form

\[
\text{Condition1 AND Condition2 AND ... AND ConditionN}
\]  \(\text{(2)}\)

and ConsequentA, ConsequentB have the form

\[
\text{Conclusion1 AND Conclusion2 AND ... AND ConclusionM}
\]  \(\text{(3)}\)

The meaning of a rule with this form is: if *Antecedent* is true, then conclude that *ConsequentA* is true, otherwise conclude that *ConsequentB* is true.

Usually rules assume the simpler IF-THEN form. Actually, some shells only support this form. Fortunately, an IF-THEN-ELSE rule may be substituted by two IF-THEN corresponding rules. For instance, the rule

Rule 1:   IF sensor A has value greater than 50

THEN water temperature is too high
ELSE water temperature is normal

may be substituted by the rules

Rule 1a: IF sensor A has value greater than 50
THEN water temperature is too high

Rule 1b: IF sensor A has value lesser than or equal 50
THEN water temperature is normal

The Antecedent of a rule may also include the disjunctive operator OR. Experience says that the use of this operator reduces readability and makes debugging harder, but it is possible to substitute a rule with a disjunctive antecedent by two rules with conjunctive antecedents. For instance, the rule

Rule 2: IF sensor A has value greater than 50
OR sensor B has value greater than 45
THEN water temperature is too high

may be substituted by the rules

Rule 2a: IF sensor A has value greater than 50
THEN water temperature is too high

Rule 2b: IF sensor B has value greater than 45
THEN water temperature is too high

It is worth noting that we may consider a rule as relating several Object-Attribute-Value triplets. Figure 8 illustrates this for Rule 1a above.

<table>
<thead>
<tr>
<th>IF</th>
<th>THEN</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
<td><strong>Sensor A</strong></td>
</tr>
<tr>
<td><strong>Attribute</strong></td>
<td><strong>has value</strong></td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>• 50</td>
</tr>
</tbody>
</table>

Figure 8 - A rule as a relation between OAV triplets

### 3.2 Inference Engine and Inference Mechanisms

In an expert system, the algorithms to process knowledge are grouped in the Inference Engine module. Its job is to process the contents of the KB, and the information coming from the UI, to infer new information. Therefore, the algorithms in the IE must use inference mechanisms capable of dealing with the knowledge representation formalisms used in the KB. For instance, when using Predicate Logic to represent knowledge, we may use Resolution to infer new information. When using Semantic Networks and/or Frames, we typically recur to Inheritance to process A-KIND-OF and IS-A relations. As Production Rules are the most used representation formalism, in this text we are going to focus on inference mechanisms to deal with them.

The basic mechanism to process rules is Rule Chaining, which consists in connecting a problem to its solution through a chain of rules. To produce the chain, the antecedent conditions of each rule are linked to matching conclusions of other rules. Inference is performed by transversing the chain. Depending on the direction of the transversal, two basic chaining methods result: Forward Chaining, from the problem to the solution, and Backward Chaining, in the opposite direction. For the sake of simplicity, in the remainder of this section we will restrict our analysis to IF-THEN rules,
with conjunctive antecedents (no OR operator) and a single conclusion. As we have already seen, this is not actually a restriction, since rules not conforming these constraints may be converted to the simpler form that we are considering.

3.2.1 Forward Chaining

Forward chaining is fact-oriented: we start with a set of Facts, and use the rules to deduce new facts. The basic algorithm is:

i) Given a set of Facts, chose a rule whose Antecedent is true, and fire the rule; the Consequent of the rule is a new Fact

ii) Use the new set of Facts to (recursively) proceed

iii) Stop when no more rules may be fired

Figure 9 illustrates the method. The initial set of facts consists of C, D and E. With this set, rule R2 is triggered, producing the new fact G (its conclusion). With the expanded set of facts, rule R5 is triggered, and the new fact H is added to the set of facts. As no more rules may be fired with the current set, the process is stopped.

![Figure 9 - Forward Chaining](image)

3.2.2 Backward Chaining

Contrasting to the former mechanism, Backward Chaining is Goal-oriented: given a hypothetical solution (a Goal), we use the rules to seek for support to that solution. A rule supports a goal when its conclusion matches it and its antecedent is true. Therefore, seeking for goal support results in searching backwards through a chain of rules. The basic algorithm is:

i) Given a Goal, stop if a Fact supports it

ii) Otherwise, chose a rule whose Consequent may produce that Goal

iii) Consider the Conditions of the Antecedent as new Goals, and (recursively) proceed

Actual shells adopt a refined version of this algorithm which assumes that the set of available Facts may be empty when the inference process starts. As a consequence, if no rule may be chosen in the second step, the algorithm asks the user about the truthfulness or the falseness of the Goal, giving rise to the characteristic dialogues of most expert systems.

The example of Figure 10 illustrates the mechanism. Given the hypothesis H, two rules may support it: R3 and R5. Let’s suppose that the algorithm chooses R3 according to some criteria. To be triggered, R3 must have a true antecedent, therefore its conditions A and F are new Goals to attain. As there is no rule which may support A, the user is questioned about its value. If the user states that it is true, A is asserted as a new Fact. As regards to F, the rule R1 is selected, and B and C are new goals. The user is questioned about their values, and if the answer is positive they are asserted as new Facts. Now, the Facts B and C may be used to fire rule R1, giving support to the goal F, which is asserted as a new Fact. From F and A, the rule R3 may be triggered, which confirms the initial Goal H as a Fact.
3.2.3 Forward vs. Backward Chaining

Forward chaining, being data-driven, tends to be more effective than backward chaining when the set of possible conditions is small, while the latter suits better to tasks with a limited set of possible conclusions.

In addition to this computational characterisation, the adoption of one of the mechanisms tends to lie on the specific characteristics of the task to perform. For instance, some tasks like monitoring are mainly data-driven (we get the data from the process, then we try to understand which process state conforms to the data), suggesting the use of forward chaining. Conversely, diagnosis tasks are typically goal driven (we hypothesise a cause, afterwards we seek for data which confirms/contradicts the hypothesis), which conforms to the fact that the great majority of expert systems for diagnosis use backward chaining. This should be understood just as a general guidance to the selection of the best mechanism to choose. It doesn’t mean that we should always adopt forward chaining when dealing with data-driven tasks, and backward chaining in goal-driven ones.

Careful analysis of real tasks show that the resort to both kinds of mechanisms, in a more or less intermixed way, may improve performance and quality of results. This has led to the adoption of mixed modes of chaining to opportunistically improve the performance of expert systems. The idea is to alternatively use both mechanisms during a problem solving session, according to a task dependent strategy. A common strategy gives priority to backward chaining, restricting the use of forward chaining to situations when it is not possible to backward chain. The converse strategy, which gives priority to forward chaining, is also commonly used.

A potentially better strategy consists in choosing one mode of chaining according to the current phase of the problem solving process. For instance, consider Monitoring and Control. Given a set of data, collected from the process, we may use forward chaining to reach to a limited set of candidate process states conforming to the available data. These candidate states may be seen as hypotheses which require further discrimination. This may be accomplished with backward chaining, which guides the acquisition of additional data (from the process, from the user, from historical data, ...) to confirm/contradict each hypothesis, and (hopefully) select only one. Given the determined process state and the overall data collected from the process, we may use forward chaining to determine the best control action to take.

3.2.4 Generating Explanations

The separate representation of knowledge in expert systems gives them the capability to easily generate explanations. The most common ones are produced as answers to “How?” and “Why?” queries. In rule-based systems, explanations are generated from the analysis of the rule chaining.

Consider the chaining in Figure 10. After reaching the conclusion H, the expert system may answer the user question “How did you choose H?” by tracing the fired rules. A possible answer is represented in Figure 11. This kind of explanation is particularly important to increase the confidence of the human user in the system, and also for debugging purposes.
4. DEALING WITH UNCERTAINTY

We may think of a rule “if E then H” as stating that the antecedent E is an evidence supporting the hypothesis H. Up to now, we have considered that E may assume the values “true” or “false”, i.e., we may establish its truthful value with absolute certainty by some means, e.g., through inference or data gathering. Also, we have considered that the evidence E would confirm hypothesis H with absolute certainty.

In many real world domains things are not that easy, and expert systems must deal with uncertainty originating in data and knowledge. Often determining E with absolute certainty is not possible or easy to attain: e.g. data collecting equipment may be inexact or inaccurate, getting all the needed data may require expensive or dangerous tests. This requires the association of a measure of uncertainty with each piece of information.

Moreover, most of the rules in a KB may represent heuristics which the expert uses to reach conclusions without complete information, i.e., with only partial evidence. A rule of this kind represents the fact that an evidence E only partially supports a hypothesis H. To deal with this, we must associate some measure of uncertainty to the rule, representing the degree of support of evidence E on hypothesis H.

Besides some form of uncertainty quantification in rules and data, a method for combining uncertainty values is also needed. Some of the approaches adopted in expert systems to deal with uncertainty are:

- Classical Probability Theory
- Bayesian Probability
- MYCIN Certainty Factors [22]
- Fuzzy Theory [23]
- Dempster-Shafer Theory of Evidence [24]
The main distinction between the different methods lies in the way how each one conceptualises and manipulate uncertainty. In this text we are going to focus on two of the most used formalisms: MYCIN Certainty Factors and Fuzzy Theory.

4.1 MYCIN Certainty Factors

In probabilistic approaches, uncertainty is viewed as the probability of a piece of information being true\(^3\). MYCIN developers adopted a different view which proved to be effective in medical applications and was also applied in many other domains: they conceptualised uncertainty as a degree of belief in a fact or rule\(^4\).

Let us denote \(cf(A)\) as the certainty factor of assertion \(A\), with values ranging from -1 to 1, \(cf(A) = 1\) meaning that there is absolute certainty that \(A\) is true, \(cf(A) = -1\) meaning that there is absolute certainty that \(A\) is false, and \(cf(A) = 0\) meaning that the truth value of \(A\) is unknown.

A MYCIN-like rule has the form

\[
\text{Name: IF Evidence THEN Hypothesis (} cf = CF_R \text{)}
\]  

where \(CF_R\), the rule’s certainty factor, denotes the belief in Hypothesis given that Evidence is observed. A rule will be fired when \(cf(\text{Evidence}) \neq 0\). When calculating \(cf(\text{Hypothesis})\), the CF of the rule serves as an attenuation factor for the certainty of the rule’s antecedent:

\[
 cf(\text{Hypothesis}) = CF_R \times cf(\text{Evidence})
\]  

When the antecedent of a rule is a conjunction, the evidences of the conditions must be combined. The computation follows the principle that “the strength of a chain is determined by its weakest link”. A threshold level \(\delta\) is also used to prevent firing rules producing Cfs close to zero, which would result in a propagation of near-zero confidence values, and therefore poorly useful information, which consequently would give rise to a loss of performance. In MYCIN, a threshold level of 0.2 was used.

Given \(E = C_1 \land \ldots \land C_n\),

\[
\text{if } cf(C_i) \geq \delta \text{ for all } i, cf(E) = \min [cf(C_1), \ldots, cf(C_n)]
\]  

\[
\text{if } cf(C_i) \leq -\delta \text{ for all } i, cf(E) = \max [cf(C_1), \ldots, cf(C_n)]
\]  

\[
\text{if } |cf(C_i)| < \delta \text{ for any } i, cf(E) = 0
\]  

\[
\text{if any two } cf(C_i) \text{ are of opposite sign, } cf(E) = 0
\]

Evidences must also be combined when two fired rules have the same conclusion (consonant rules). The computation is performed using a set of heuristic formulas.

Given two rules with the same conclusion \(H\), let \(cf1\) and \(cf2\) be the certainty factors computed for \(H\) with each one of the rules. The resulting CF for \(H\) is:

\[
\text{if } cf1 > 0 \text{ and } cf2 > 0, cf(H) = cf1 + cf2 - cf1 \times cf2
\]  

\[
\text{if } cf1 < 0 \text{ and } cf2 < 0, cf(H) = cf1 + cf2 + cf1 \times cf2
\]  

\[
\text{if } -1 < cf1 \times cf2 < 0, cf(H) = (cf1 + cf2) / [1 - \min(|cf1|, |cf2|)]
\]  

---

\(^3\) For a detailed analysis of probabilistic approaches, their advantages and drawbacks, see for instance [21], Chapter 4.

\(^4\) Actually, in MYCIN the certainty factor was a composite of a measure of belief MB and a measure of disbelief MD. In this paper, for the sake of conciseness, we adopt the simplified account of the methodology described in [18].
if \( cf1 \times cf2 = -1 \), \( cf(H) = 0 \)  \( (13) \)

These formulas may be generalised for any number of consonant rules by applying them to two rules at a time. Note that, by (10) and (11), when multiple rules jointly confirm (disconfirm) a same hypothesis, its confidence asymptotically approaches 1 (-1). Also, if a single rule produces a confidence of 1, the combined confidence will be 1, except if a contradictory result of -1 is produced by another rule.

4.1.1 Example

Consider the next set of rules:

- R1: IF A=i AND B=j THEN X=x \((cf = 0.8)\)
- R2: IF A=k AND B=l THEN X=y \((cf = 0.8)\)
- R3: IF C=m AND D=n THEN X=x \((cf = 0.9)\)
- R4: IF C=o AND D=p THEN X=y \((cf = 0.4)\)

Suppose that the following certainty factors are known:

- \( cf(A=i) = 0.6 \)
- \( cf(A=k) = 0.4 \)
- \( cf(C=m) = 0.4 \)
- \( cf(C=o) = -0.3 \)
- \( cf(B=j) = 0.5 \)
- \( cf(B=l) = 0.9 \)
- \( cf(D=n) = 0.5 \)
- \( cf(D=p) = -0.8 \)

Applying formulas (5), (6) and (7), we get the CFs computed from each rule:

- \( cf(X=x)_{R1} = \text{min}(0.6, 0.5) \times 0.8 = 0.5 \times 0.8 = 0.4 \)
- \( cf(X=y)_{R2} = \text{min}(0.4, 0.9) \times 0.8 = 0.4 \times 0.8 = 0.32 \)
- \( cf(X=x)_{R3} = \text{min}(0.4, 0.5) \times 0.9 = 0.4 \times 0.9 = 0.36 \)
- \( cf(X=y)_{R4} = \text{max}(-0.3, -0.8) \times 0.4 = -0.3 \times 0.4 = -0.12 \)

From these results and formulas (10) and (12), we get:

- \( cf(X=x) = 0.4 + 0.36 - 0.4 \times 0.36 = 0.76 - 0.14 = 0.62 \)
- \( cf(X=y) = (0.32 - 0.12) / [1 - \text{min}(0.32, 0.12)] = 0.2 / 0.88 = 0.28 \)

We may conclude that the value for the attribute X with a greater degree of belief is “x”.

4.2 Fuzzy Theory

Fuzzy Theory is easy to understand in terms of Set Theory. Classical sets may be discrete, when members take values from a discrete set (e.g., the set of models of cars with ABS made by a given manufacturer), or continuous, when members take values from a continuous set (e.g., the set of positive reals above 10). We may represent a continuous set through a membership function (Figure 13) which assumes the value 1 for members of the set and the value 0 for non-members.

Suppose we want to represent the set of small numbers. If we assume that a number is small if it is lesser than a given number, let’s say 6, we obtain the membership function of Figure 14.

---

5 The figures and the example in this section were adapted from [25].
This representation clearly has a problem: even if we consider that the number 6 is a good reference to distinguish between small and non-small numbers (which is arguable and depends on the context where we are working), how can we accept that 5.99 is a small number and 6.01 is not? The discontinuity we observe in the function is obviously not appropriate to represent fuzzy concepts like “small”, “tall”, “too hot” or “has fever”. The point is: membership to the set of small numbers must be defined in a continuum, each number belonging to the set in a certain degree, or with a certain likelihood. The membership function for “small numbers” in Figure 15 is clearly a better representation for the set.

In fuzzy sets, membership is defined by a continuous function with values ranging from 0 to 1. From this point of view, they may be considered as a generalisation of conventional sets. They are used to represent concepts whose definition is uncertain, and offer the possibility of using linguistic variables and qualifiers which commonly appear in expert rules.

Similarly to what happens with conventional logic, fuzzy logical operations may be performed by means of set operations (Figure 16): logical conjunction corresponds to set intersection, and disjunction to set union; negation of a membership function is computed by its complement to 1.
We also may use fuzzy rules, which represent the mapping between the membership functions of the antecedent and consequent. While in conventional rules we may infer that the consequent is true when the antecedent is also true, because the membership functions have just the values 1 or 0, in fuzzy rules the likelihood of the consequent depends on the degree of membership of the antecedent value to the antecedent fuzzy set.

Figure 17 illustrates how inference is performed. In this example, the sets for “the weather is warm” and “the beach is full” are graphically represented. If the temperature is known to be 27.5°C, we may say that the degree of membership of this temperature to “the weather is warm” is 0.3. Using the rule “IF the weather is warm THEN the beach is full”, we may infer that the likelihood of “the beach is full” by mapping the antecedent degree of membership to the consequent set, which gives a result of 0.4.

The next example illustrates the use of fuzzy sets in a simple control application. We will use the example to introduce other important concepts of the fuzzy theory.

4.2.1 Example

Suppose we need to control the water temperature of a home shower by adjusting the mixing faucet. The values for the state variable “water temperature” are percentages of the whole range of temperature, and the values for the control variable “faucet opening” range from -2.0 (maximum cold water) to +2.0 (maximum hot water).

First of all, we must decide the vocabulary (qualifiers) to use for the two variables involved:
Table 1
Qualifiers for the fuzzy variables

<table>
<thead>
<tr>
<th>Values for water temperature</th>
<th>Values for the faucet opening</th>
</tr>
</thead>
<tbody>
<tr>
<td>freezing</td>
<td>much more cold water</td>
</tr>
<tr>
<td>cold</td>
<td>more cold water</td>
</tr>
<tr>
<td>warm</td>
<td>don’t move</td>
</tr>
<tr>
<td>hot</td>
<td>more hot water</td>
</tr>
<tr>
<td>scalding</td>
<td>much more hot water</td>
</tr>
</tbody>
</table>

Next, we must define the fuzzy sets for the qualifiers. The membership functions for the qualifiers of “water temperature” are represented in Figure 18, and those for “faucet opening” in Figure 19.

![Figure 18 - Fuzzy sets for “water temperature” qualifiers](image1)

![Figure 19 - Fuzzy sets for “faucet opening” qualifiers](image2)

Now we may define a set of fuzzy rules:

R1: IF “water temperature” = “freezing”
    THEN “Faucet opening” = “much more hot water”

R2: IF “water temperature” = “cold”
    THEN “Faucet opening” = “more hot water”

R3: IF “water temperature” = “warm”
    THEN “Faucet opening” = “don’t move”

R4: IF “water temperature” = “hot”
    THEN “Faucet opening” = “more cold water”

R5: IF “water temperature” = “scalding”
    THEN “Faucet opening” = “much more cold water”
Suppose that the user classifies the water temperature as “almost good” and we are able to map by some means this qualifier to 40% of the temperature range (see Figure 20). Alternatively, you may suppose that the input value was directly measured.

As we may see in the figure, the degree of membership of this temperature is 0.68 for the set “good”, 0.32 for the set “cold” and zero for the remainder. The process of converting a real number to a set of degrees of membership to fuzzy sets is called Fuzzyification. From the rules R2 and R3, we may map these values to the “more hot water” and “don’t move” fuzzy sets. The resulting consequent’s likelihood is represented in Figure 21.

Now the problem is confined to how to decide the control action to take: we must convert the output of the inference in a non-fuzzy (crisp) value, i.e., a real number. This is called Defuzzification, and there are several methods to do it. One possibility is to choose the action with maximum likelihood. In this case (see Figure 22) there is a maximizing interval, therefore we chose the median of the maximizing set, which is 0. This is called Mean-of-Maximum Method, or simply Maximum Method.

An alternative is the Moments (or Center-of-Area) Method, which consists in computing the center of the area of the likelihood function. In this case (see Figure 22), the result is 0.3, which corresponds to slightly opening the hot water\(^6\).

\(^6\)For a more detailed view of defuzzification methods, including others not mentioned in this text, see [26], Section 6.3.
The Maximum Method is particularly suitable for applications which require a decision from a set of discrete values (e.g., “buy gold” vs. “buy stocks”). Note that with the rules and sets of this example, if we make the input value change continuously over its range the output value will exhibit discontinuities and will assume values over a discrete range when we use this method. Conversely, with the Moments Method the output behaviour will change continuously, which makes the method more appropriate for control applications. This method has, however, some drawbacks: besides being computationally expensive, there are situations where it produces unsuitable results, because the center of the area may be a value with, let’s say, zero likelihood.

5. DEVELOPING EXPERT SYSTEMS

5.1 The Life-Cycle of an Expert System

Knowledge Engineering is the branch of AI which deals with the expert systems development process. Among the activities of a Knowledge Engineer we may mention: knowledge acquisition; assessment of the suitability of knowledge-engineering technology for a prospective application; planning and management of an ES project; design of appropriate representation formalisms; design of inference and control mechanisms; implementation or selection of a suitable ES shell; design and implementation of a KB; integration of an ES with its operating environment; evaluation of an ES.

The most difficult and tricky of them is Knowledge Acquisition (KA), for the reasons we have already pointed out in Section 2.4. Moreover, the other activities have identical counterparts in general Software Engineering, contrarily to KA, which is characteristic of knowledge-based systems.

Many methodologies have been proposed to guide KA, and consensus is far from being reached among researchers and practitioners in the area. In this text we present an approach\(^7\) which we have applied with satisfactory results in the development of many small-to-medium size ES.

Under this approach, the life-cycle of an expert system is represented in Figure 23. There are many similarities with the life-cycle of other software products: we find the same four phases in many Software Engineering methodologies, and the same happens with many of the Steps. Nevertheless, the obstacles associated to KA have led to the adoption of two development techniques which, although being used in general software engineering, assume a key role in knowledge engineering: cyclic development and early prototyping.

As we may see in Figure 23, cyclic development may actually embrace the development and deployment phases, and involves two types of cycles: Extension Cycles, intended to broad the scope\(^8\) and capabilities of the ES, and Refinement Cycles, intended to increase the knowledge accuracy. With this approach the knowledge engineer may focus his KA activities in small amounts of knowledge in each cycle, which facilitates his communication with the expert and enables a solid deepening of the acquired knowledge.

The first cycle is obviously of the first type, and produces an initial working prototype. Although limited in scope, this prototype usually represents a key factor in the success of the ES overall development: first, because it may determine the enthusiasm of the expert, which often discovers that “it works” when faced with the prototype; second, because frequently the financial provision for the overall project is assured only when a working prototype is shown to the potential supporters.

In subsequent iterations, the prototype is gradually refined or extended in scope and capabilities. This evolving prototype constitutes a very important KA support tool, as it favours

\(^7\) In this Section we follow the methodology presented in [27]. Also, the Figures of the Section are reproduced from this book.

\(^8\) The scope of the ES is viewed as the extent of the task that it may perform and the range of situations in which it can perform the task [27].
communication with the expert and may function as a testbed for the validity and completeness of the acquired knowledge.

The Analysis Phase includes the Identification of a potential application, taking into account possible motivations for the development of an ES and the characteristics of the task, and also the Assessment of the suitability of the ES technology, which comprises the evaluation of technical feasibility, of the balance between benefits and costs, as well as organisational, social, legal and other implications.

The Specification Phase is devoted to clearly define what the ES will do. It comprises Familiarisation with the expert’s task, to get a general overview of it, with the aim at specifying the overall functionality of the ES, and decomposing it to produce a plan for the incremental development.

The Conceptual Design is the first step of the Development Phase, and consists in acquiring expert knowledge for the current scope, focusing on the data needed to solve the task, how the expert infers new information, and the sequence of steps the expert performs to accomplish the task. The result of this step is an organised body of knowledge, the Conceptual Model. The Implementation Design intends to specify how the conceptual model will be implemented with the selected ES shell, while Implementation is intended to create a working ES according to the implementation design. In advanced cycles of development, the implementation stage may require the integration of the ES into its operating environment. The last step of Development is Evaluation. Roughly, in early cycles of development this involves a simple "unit test" of the added functionality, whereas in cycles preceding deployment a formal validation of the overall performance of the ES is done, and the results indicate whether the ES is ready to be fielded.

The Deployment Phase embodies Fielding and Maintenance. Fielding may include final integration with other systems (e.g., equipment providing input from sensors), installation of specific equipment (e.g., the hardware in which the ES will run) and users training when appropriate. The goal of Maintenance is keeping the ES usable and used, which may involve routine user support, installing new versions of the ES and starting new refinement and extension cycles when needed.

![Figure 23 - Expert Systems life-cycle](image-url)
5.2 Knowledge Acquisition

We may verify in Figure 23 that there is no Knowledge Acquisition phase or step in the Life-Cycle of an ES. This is because KA activities play a role in each step in the system’s life-cycle, as we may see in Figure 24.

![Figure 24 - Typical distribution of percentage of time spent with KA](image)

We may identify two stages in KA activities (Figure 25): Initial Inquiry, which runs during Analysis and Specification, and Detailed Investigation, during Development and Deployment. In the Initial Inquiry, the goal of the Knowledge Engineer is to get a broad and shallow overview of the task, in order to specify the ES and plan its development. Detailed Investigation is characterised by focus on detail, to get a deep perspective of what the expert knows and how he performs the task. We are going to take a closer look at each of these stages.

![Figure 25 - Knowledge acquisition stages](image)

5.2.1 Initial Inquiry

This stage lays the foundation for the cyclic development. The knowledge engineer works with the expert, the intended users and the project funders to define the specification of the ES and plan the iterative development.

The first question to answer is: what will the ES do? This involves:
i) the definition of the goals which the introduction of the ES is expected to accomplish

ii) the definition of the functionalities it must exhibit

iii) the identification of the users

iv) the specification of the operating environment

v) the identification of the sources of expertise

Once this is clarified, the knowledge engineer can start working with the expert to get a broad understanding of how he performs his task. Depending on the particular situation, this can be accomplished through the direct observation of the expert’s activity, meetings where the expert explains the way he performs his task, and/or discussion meetings around specific cases. It is expected that the knowledge engineer ends familiar with all aspects of the task and with the full range of situations in which the task will be performed, without losing focus with unnecessary details: KA will concentrate on deepening knowledge during the second stage.

This broadly acquired knowledge will serve as basis for subsequent discussions with the expert, focused on the elaboration of a development plan. The first thing to do is to define possible partitions for the overall scope of the ES. Partitions may be determined along three dimensions:

- by dividing the task in subtasks (e.g., monitoring, control, diagnosis, elaboration of reports)
- by dividing the input data into a set of classes (e.g., data from ProcessA, data from ProcessB)
- by dividing the output data into a set of classes (e.g., control actions for ProcessA, control actions for ProcessB, diagnoses, reports)

Partitions must then be compared according to

- size (amount of knowledge to acquire)
- difficulty (of the task to perform)
- importance (how significant is the partition to the overall goals of the ES)

The results of this comparison are then used for the identification of the development increments, i.e., the partitions that will actually be considered for plan elaboration and will represent the functionalities to be added in each development cycle.

Once the increments are identified, they are ordered to form the plan, according to the results of the comparison. In principle, a higher priority is given to the partitions with less size, less difficulty and more importance. When appropriate, the funders or their representatives may be asked to participate in this process.

To complete the plan, the development time of each increment must be estimated.

5.2.2 Detailed Investigation

In each cycle of this stage, the KA work is centered around a limited set of functionalities, the current scope, according to the development plan. The sequence of activities which are to be performed is:

i) Acquire knowledge for the current scope

ii) Organise knowledge

iii) Create/update conceptual model

iv) Integrate knowledge in KB (not a KA-specific activity)

v) Evaluate system's performance
Actual knowledge acquisition comprises working with the expert to precisely understand all the details and facets of his problem-solving activity. To help this work, it is advantageous to consider different kinds of knowledge and focus each KA session in one of the kinds. A convenient knowledge classification for this purpose is:

i) Strategic Knowledge, which describes the steps taken by the expert when performing the task

ii) Inferential (or judgmental) Knowledge, which describes how the expert deduces new facts and hypothesis from the available information

iii) Factual Knowledge, which describes the attributes used by the expert when performing the task

Acquiring strategic knowledge requires a deep understanding of the task steps, which may be attained through detailed task descriptions and detailed discussions around specific cases, focusing on strategy.

Inferential knowledge requires the identification of each reasoning step performed by the expert. Detailed case discussions focusing on inference are valuable means for this purpose. Particular attention should be devoted to the use of uncertainty, and to find hidden assumptions, i.e., factors which the expert neglect to mention because they consider them as implicit or obvious.

Acquiring factual knowledge comprises the identification of attributes and the definition of their ranges of values. Input data involved in the reasoning steps, as well as final and intermediate conclusions, may constitute the starting points for this work. Sessions devoted to factual knowledge may involve reviewing and discussing specific cases, as well as notes from sessions devoted to inferential knowledge.

The collected knowledge must now be conveniently organised to facilitate the design and the implementation of the ES. A thing to do is to organise the records of interviews and meetings to enable a quick access to every peace of information (e.g., creating indexes organised by topic). Also, a Project Dictionary of standard terminology used by the development team should be created. This document is important to guarantee the consistency of the terminology and concepts among the team members, and must be kept up-to-date during all the development. Terms from the standard project vocabulary and terms with non-obvious definition should be included in the dictionary. Other document to create and keep up-to-date is a Case Library containing the collection of case reports, which are written accounts of the intended behaviour of the ES for each specific relevant case discussed with the expert. Besides being an important help to KA, this library may also be used later during evaluation.

After organising the acquired knowledge, there is the need to do a careful analysis of each piece of knowledge, to clarify its function within the knowledge base. The result of the analysis will be a Conceptual Model, which is a set of intermediate representations for the three kinds of knowledge: strategic, inferential and factual. Intermediate representations are notations and diagrams which allow the representation of each piece of knowledge according to its function, and are independent of the specific representation formalisms available within the ES shell adopted for implementation. Examples of intermediate representations for strategic knowledge are: flow charts, functional-decomposition trees and decision trees. For inferential knowledge, we may use, for instance: decision tables, rules, decision trees and inference networks. Examples of intermediate representations for factual knowledge are: fact tables, taxonomic trees and graphs9.

The conceptual model provides an intermediate means for representing knowledge which is close to the human being and to the machine: the intermediate representations are easily understood by the expert and the knowledge engineer, and are easily translated to any Shell-specific

---

9 Illustrations of the use of these representations may be seen at [27], Chapter 9.
representation formalism. Therefore, the conceptual model constitutes a key tool in the design and maintenance of the ES as it may support both knowledge acquisition and KB implementation.

In which concerns to KA, it serves as a communication medium with the expert, and helps establishing context for discussions. It also supports implementation as it helps the analysis of the requirements for the ES Shell, and serves as specification for the KB.

After the implementation of the KB, an evaluation of the system's performance must be done. Possible evaluators are:

i) the project expert

ii) funders' representatives

iii) potential users

iv) experts not associated to the project

Evaluation sessions may be focused on two distinct aspects of the system's performance:

i) the results, i.e., the correctness of the ES problem-solving behaviour;

ii) the interaction, i.e., the way in which the ES interacts with the user.

There are two usual forms of evaluation sessions:

i) the demonstration session, where one or more evaluators review the system while it performs a task;

ii) the off-line session, where one or more evaluators review a written record of an ES problem-solving session.

Although this must not be taken as a definitive rule, demonstration sessions are typically more effective when evaluating the interaction, while off-line sessions are mostly used to evaluate results.

6. EXPERT SYSTEMS FOR MONITORING AND CONTROL

Expert systems, offering the possibility of automating the use of knowledge from the best experts for solving problems, have emerged as very promising tools at mid-seventies, and progressively evolved to a mature state during the eighties, when they have left research labs to tackle industrial and commercial applications. Nowadays, ES are found in a broad diversity of fields, from business to science, engineering, and manufacturing. The arising of low-cost commercial Shells running on personal computers and offering advanced integration and interface capabilities with databases, spreadsheets, data-gathering applications and other conventional programs, favoured the spreading of practical applications of ES. A recent work on the evolution of the field indicated an impressive growth, as researchers developed systems to deal with difficult but commercially rewarding problems [28]. The work was based on a survey [29] which revealed that diagnosis is the most frequent type of application problem for ES, and monitoring and control jointly occupy an important position (Figure 26).
During the last decade, expert systems for control applications were developed and/or fielded in a wide range of domains. One of the first systems referred in the literature was ESCORT [2], whose prototype was tested on a simulation of an oil platform process plant. PICON [13] was the first commercially available development environment for real-time expert systems. G2 [30], derived from PICON and developed by Gensym Corp., is a commercial ES Shell specially tailored to real-time control applications which is currently being applied to several problems. Literature also refers Pilot’s Associate [31], intended to give support to the piloting of battle-planes. The RESCU project [32] produced a quality control system. LES [33] was developed to monitor and diagnose the Space Shuttle’s launch processing system. IDEA [34] is being currently used in 1,500 Fiat/Lancia/Alfa Romeo repair centers throughout Italy and helps diagnose faults in a variety of automotive electronic subsystems. Sparse [35] is intended to process alarms of power systems. Other examples are MCM [36], DECA [3], IPCS [14], L*STAR [37], PISCES [38], DANTES [39] and AMPERES [40]).

Given the severe requirements for developing expert systems for monitoring, diagnosis and control, particularly when hard real-time constraints are present, as described in Section 2.3, the first fielded applications resorted to high-performance workstations with specially tailored architectures (e.g., PICON). Presently, the technological evolution of personal computers, the growing flexibility, interfacing capabilities and performance of the ES Shells are changing the scenario. Expert systems for process monitoring, diagnosis and control are reaching interesting success levels [41].

Current trends indicate a growing integration of ES with conventional control systems, giving rise to highly autonomous systems, the expert controllers, with the ability to incorporate and represent the nonlinearities needed for dynamic systems [42]. Approaches which entirely embed the ES in the control equipment are being explored (e.g., [43]).

One of the known problems of expert systems is that they exhibit a brittle behaviour when they rely solely on heuristic knowledge. The lack of a deeper body of knowledge makes difficult to deal with unforeseen situations. Moreover, as we have already seen, expert knowledge acquisition is a hard task, subjected to errors and misunderstandings; relying exclusively on the expert’s knowledge may be acceptable for a great number of situations, but may be too risky for critical applications. Also, explanations generated exclusively from heuristics are often too superficial. To overcome these problems, some of the expert systems presented above embody explicit models of the process under control, and use both forms of knowledge (heuristic and model-based) to solve problems and generate explanations. Explicit models may also facilitate heuristic knowledge acquisition and KB long-term maintenance.

---

10 Reproduced from [28].
Other alternative AI techniques are being applied to intelligent monitoring and control. Artificial neural networks, which crudely emulate biological neural networks, exhibit fast response times and learning capabilities. They may be used to improve the control system behaviour by learning on-line [42]. Genetic algorithms are being used to evolve controllers off-line [42].

7. CONCLUSION

Expert systems represent a viable approach to knowledge-intensive problem solving tasks. Their application to monitoring and control may relieve the cognitive overload to which the operators are exposed, which may improve system’s performance and safety. A careful assessment of the situation should however been accomplished before starting construction of an expert system, as knowledge acquisition is tricky and time-consuming. Special attention should be devoted to time-response, interface and integration with the control equipment, and availability of expert(s) to allocate to the project.

REFERENCES


BIBLIOGRAPHY


Constantin V. Altrock, Fuzzy Logic and Neurofuzzy Applications Explained (Prentice-Hall, 1995).

ADDITIONAL READINGS


APPLICATION OF EXPERT SYSTEMS IN HIGH ENERGY PHYSICS: 
THE ALEPH AND ZEUS CASE

P. Mato
CERN, Geneva, Switzerland

Abstract

Two cases in which Expert System techniques have been used for the operation of large high energy physics experiments are presented. In both cases, the driving force has been to achieve better performance by increasing the efficiency and reliability of the online system. The scope and implementation of both systems will be described and the differences of the two approaches will be highlighted. Finally, a summary of the lessons learned by the deployment and use of these Artificial Intelligence methods in running a complex experiment will be given.

1. INTRODUCTION

The objective of this lecture is to illustrate, using two examples, the way that Artificial Intelligence (AI) techniques, in particular Expert Systems, have been used to assist shift crews in the operation of currently running High Energy Physics (HEP) experiments. These two examples are ALEPH and ZEUS. ALEPH is one of the four experiments at the LEP collider at CERN, and ZEUS is an experiment located at HERA, the electron-proton colliding facility of DESY. The deployment of these Artificial Intelligence (AI) techniques required solving practical problems. The experience gained can perhaps be used during the design of the new generation of DAQ systems for experiments that are under preparation.

The lecture comprises four parts. The first part is a general review of what motivated the two experiments to start Expert System projects. The second and third parts are the reviews in some detail of the ALEPH and ZEUS expert systems: DEXPERT and ZEX. For each one, the analysis of the particular requirements, the design chosen and the way it has been implemented is reviewed. Finally, in the last part, we look to the future and try to see what can lessons we have learned that may be applied to the new generation of experiments.

2. MOTIVATIONS FOR EXPERT SYSTEMS

Both ALEPH and ZEUS collaborations decided at a given moment to launch an expert system project. The goals established by both collaborations were to increase the efficiency and reliability of the operation of the experiment, and to allow a reduction of the manpower needs and level of expertise of the shift crew. The efficiency can be increased if errors or anomalous situations are automatically handled and recovered. This is especially true if the recovery is done in less time than it would take an average trained shifter to perform the recovery manually. Concerning the reduction of manpower in the particular case of ALEPH, it was decided by the collaboration to run the experiment with two people in the control room without a DAQ expert on shift.

Another motivation was to study the applicability of Artificial Intelligence (AI) techniques, in particular of expert systems, in HEP experiments. It was also important in the case of ZEUS to see how computer science theories like pattern recognition and graph grammars could be applied to solve practical problems in running experiments.
2.1 Handling Complexity

The ALEPH and ZEUS experiments are representative examples of large HEP experiments of the early 90’s. They consist of a large detector with about ten sub-detectors, hundreds of physicists, a large DAQ system with hundreds of crates and processors, etc. The diversity of hardware and software components is also something that is remarkable. For example, the number of different programs that are running concurrently during data-taking to perform the various functions is of the order of a hundred. The interactions between the various sub-systems like DAQ, trigger and timing, detector control, data monitoring, safety, etc. are also complex. An anomalous behaviour of one element in one sub-system may affect in a non-trivial manner other sub-systems.

These experiments have been in operation for unprecedented periods. During their lifetime, the DAQ system and the environment have been in continuous change. Addition of new sub-detectors, changes in hardware or software, upgrades of the operating system, etc. It is clear that nobody can be an expert of everything. Knowledge is distributed and unfortunately knowledge is evaporating each time people in charge of parts of the experiment are replaced. In addition, physicists who are not in general experts on the trigger, DAQ, detector, etc., are operating these experiments.

2.2 Improving Performance

The first thing we need to do before we try to improve the performance is to measure it. The efficiency of the Trigger and DAQ system is measured by the ratio of interesting physics events collected and stored to the number of events produced by the accelerator. This efficiency is expressed as the product of various efficiency factors weighted by the luminosity of the machine. The DAQ efficiency factor is the fraction of the time the DAQ system is operational. To improve performance we need to minimise the time the Trigger and DAQ system is not operational. Expert systems may play a role in that since they can provide diagnosis and recovery of problems faster than a human operator can.

3. DEXPERT

3.1 Scope and Requirements

The operator controlling the ALEPH DAQ system is in charge of performing the start and stop sequences and also of handling the error conditions that may occur during data-taking. The origins of these errors are violations of the protocols that dictate the behaviour of the different components of the system. These protocols maybe violated because either hardware or software malfunctions. Upon detection of an error, any task in the system can force the run controller to go into an error state, automatically disabling the trigger. At this moment the operator has to diagnose the error and apply the corrective actions to resume data-taking. DEXPERT (DAQ Expert System) was developed to assist the operator with the recovery from read-out errors.

The main requirement for DEXPERT is that it should emulate as closely as possible the behaviour of a human expert operating the ALEPH DAQ system. As the human operator does, DEXPERT should react automatically to read-out errors, apply its knowledge about this particular problem domain, receive error messages, access databases, perform corrective actions using the same control programs as the operator, and finally, restore the running conditions (see Figure 1). DEXPERT gets interrupted when an alarm or error message is received from the general ALEPH Error Logger which is in charge of collecting and logging all the errors, which are produced by any part of the system. It is not the role of DEXPERT to monitor the system and detect the anomalies. The error detection is done at the source. It should also be possible for the operator to enable/disable DEXPERT like an autopilot and intervene if it goes out of control.

The constraints for DEXPERT included that it should be integrated within the existing ALEPH DAQ system. This implies using the ALEPH standard packages for communication, user interface, database access, etc.
3.2 Design and Implementation

DEXPERT was designed with two well-differentiated parts: the thinking part and the interface part. An overview of the design is shown in Figure 2.

- The thinking part, the *Brain*, is where the problem is analysed and decisions are taken. This part is implemented using an expert system tool.

- The interface part is composed of a number of independent objects called *Tentacles*. Each *tentacle* is specialised to interact with a specific component of the system (i.e. one knows how to talk to the *run controller*, to the trigger controller, to the operator, etc.). In addition a more general object called the *Cerebellum* that serves as a bridge between the *Tentacles* and the *Brain* is also needed.
DEXPERT can reason about three basic data types: Alarms, Actions and Action replies (see Figure 3).

- The Alarms are generated by the Tentacles after reception of external or internal stimuli. They are collected by the Cerebellum which queues them to the Brain triggering the start of reasoning.

- The Actions are the results of the reasoning process of the Brain. They are queued to the Cerebellum who dispatches them to the Tentacle that is in charge of ensuring that that particular action is executed.

- The Action replies of the scheduled Actions are collected by the corresponding Tentacle and are sent back to the Brain through the Cerebellum to allow reasoning to resume.

![Diagram showing basic data types being exchanged between DEXPERT components](image)

Each of the main components of DEXPERT is modelled as a finite state machine (FSM). Figure 4 shows the FSMs for the Cerebellum and Tentacle. Each component can proceed in an asynchronous way as if they had a life of their own, allowing DEXPERT to be able to execute several actions at the same time.

![Diagram showing finite state machine for Cerebellum and Tentacle](image)

DEXPERT is implemented as a VMS process running in one of the ALEPH on-line computers. The Cerebellum and Tentacle components and the basic data types are implemented as C++ classes. The Brain is implemented as a C++ class wrapper to a rule-base production system written in the OPS5 [4] language. In the following sections we will go into more in depth the way in which the Brain has been implemented. Starting with the criteria for selecting such a tool, what the tool consists of and the kind of expertise we have been able to program in the Brain.

### 3.2.1 DEXPERT Brain

One of the main practical requirements for selecting what expert system tool to choose in order to implement the Brain was that the tool must be callable from outside (start thinking when errors and
alarms have been collected) and also have the possibility to call external functions which are needed when accessing databases, performing actions, etc. The tool should support forward chaining since in this application we do not need to know the exact origin of the symptoms in order to apply the recovery actions. From facts we deduce new facts and apply recovery actions.

The OPS5 programming language was selected since it fulfils the requirements and also it is well integrated in to the VMS operating system. OPS5 is a powerful pattern-matching language developed at Carnegie Mellon in the late 70’s. It has been used to develop large industrial knowledge-base systems. An example of a production rule that is used in DEXPERT is shown in Figure 5.

```
(P TRIGGER_ERROR::TMO_Wait_No_Busy
 ( <MODULE> ( MODULE ^NAME TRIGGERERROR
 ^LEVEL <L>
 ^STATE GO )
 ( <ALARM> ( ALARM ^ERRORNAME TRIGGERERROR
 ^P1 [TMO_Wait_No_Busy]
 ^P2 <P2>
 ^P3 { <P3> <> MANY } ) )

-->
 ( CALL BRAIN_GET_INFO FIOD <P2> <P3> )
 ( CALL BRAIN_ERRMSG [Trigger_Error::TMO_Wait_No_Busy>> Doing a FIOD] )
 ( MAKE TRIGGER_ERROR::HANDLE_BUSY_TMO )
)
```

Figure 5 Example of a production rule used in DEXPERT

The expertise of DEXPERT can be classified in to 3 types: Heuristic knowledge, decision trees and recovery sequences. Most of the knowledge is of the heuristic type. Very often it does not need to know the real cause of the problem (full diagnosis) to be able to execute the proper recovery actions. The rules of the DEXPERT production system are chained to produce decision trees to diagnose sufficiently the problem up to the level of being able to select the proper recovery. Usually the recovery of a problem requires a sequence of actions to several parts of the system. These sequences are also part of the knowledge base.

There are about 250 rules in DEXPERT. Due to some limitations in OPS5 that considers any rule at the same level as any other, there was the need to put some effort in to managing these rules in a more modular way. An example of that is the set of rules needed for sequencing the actions for a given recovery. This set of rules could be called from various decision trees and the executed sequences should not get mixed.

### 3.3 Operating DEXPERT: Successes and Failures

The development of DEXPERT was done fast and quickly put in production. It is able to handle about 90-95% of the possible errors during data-taking. The difficult problems for which there is not a well established recovery are simply not handled and given up passing the control to the human operator.

DEXPERT was extremely useful during the first years of running the ALEPH experiment, since the number the errors was higher than now (presently there are less “bugs” in the software and better hardware). It clearly fulfilled its original goals (increased efficiency and allowed running without a DAQ expert on shift).

One of the problems that ALEPH encounters now is that the average shift crew is less knowledgeable of the system and relies heavily on the expert system solving the problem. Also some
sort of failure is that the usage of these expert system techniques has not been extended to other parts of the system like the slow controls. However, the big problem is the difficulty to maintain the knowledge base. Only experts knowing the OPS5 language can do this. The language is complicated and intrinsically difficult to debug. Ideally, an interface could have been built to enter the “expert knowledge” in an easy way, i.e. using a graphical user interface, and then producing automatically OPS5 code which then could be compiled into DEXPERT. This kind of interface program was never realised.

4. ZEX

4.1 Scope and Requirements

The ZEUS collaboration decided in 1992 to construct an expert system to support the operating of the experiment. ZEX stands for ZEUS Expert System. The goals were very similar to those of ALEPH. Firstly to increase the efficiency and reliability of the experiment and secondly to store the knowledge of various real experts of the experiment and to make it available to everyone. The project was divided into several stages. The first stage was the development of the ZEX prototype initiated at the end of 1992 and put into operation in 1993. The ZEX prototype was used for diagnosing pre-selected aspects of the experiment data transmission. Based on this experience, the development of the extended system (ZEX) covering all key areas of the experiment was started in 1994 and put into production in 1996.

ZEX covers more areas of the experiment than DEXPERT. In particular the “slow controls” are covered by ZEX and not by DEXPERT. Concerning the phases in the processing of errors: (a) monitoring the DAQ system for symptoms of anomalous behaviour, (b) finding the origin of this anomalous behaviour (diagnosis) and (c) recovering the DAQ system from errors, ZEX focusses primarily on (a) and (b) while DEXPERT focuses on (b) and (c).

The system interfaces, which embed ZEX into the online DAQ system, are shown in Figure 6. Input to the expert system comes from the various sub-systems, output is to the DAQ system. The knowledge of several experts is required to prepare and tune ZEX. Experts from the different components provide detailed knowledge about the monitored quantities. A knowledge engineer prepares the routines for the general purpose analysis, and the DAQ coordinator collects knowledge about case specific treatments.

![Figure 6 ZEX system environment](image-url)
4.2 Design and Implementation

ZEX has been designed using OO methodology. The design model allows one to define the knowledge base of any expert system in accordance with basic OO principles of abstraction, encapsulation, modularity and hierarchy. This methodology has been applied to both parts of the knowledge base: entity-level knowledge (solution space) containing such objects as input data, partial solutions, final solutions and control data, and problem-solving knowledge that is the set of interpretative procedures used for reasoning over data in the solution space.

An approach called Blackboard has been used to design ZEX. In this approach the system is partitioned into (see Figure 7):

- A global hierarchical data structure of a solution space called Blackboard.
- Independent hierarchically organised Knowledge-Sources (KS) containing problem-solving knowledge, which run under the controller.

The entity-level knowledge of ZEX is stored in the Blackboard according to a hierarchical structure (system → subsystems → components)

![Blackboard Architecture Schema for ZEX](image)

The problem-solving knowledge (interpretative procedures) in ZEX is modularised and encapsulated in Knowledge-Sources (KS). Each KS can be either Rule-based knowledge (production system) or Patter Recogniser-based knowledge as shown in Figure 8.
4.2.1 ZEX Expertise

ZEX is a hybrid expert system. Two different techniques are used together to express the problem-solving knowledge in ZEX: Syntactic pattern recognisers and rule-base production systems. The first is mainly used to detect the symptoms of anomalous behaviour and the later to perform the diagnosis of the problem.

A production system consists of a working memory containing assertions (statements that certain facts are true) that correspond to input data and intermediate results, a set of rules of the form “if <condition> then <actions>”, and a rule interpreter or inference machine that evaluates the rules if any assertion changes in the condition part. Conflict resolution strategies are needed to select the order in which rules are executed in case of conflict. In case of real-time applications most assertions are treated as functions of time, i.e. it is necessary to consider the present values of process data as well as the past ones.

The production system of ZEX has been implemented using the RTworks shell from Talarian. RTworks is a family of software products for building client/server applications that intelligently manage time-critical data.

A rule-based system is not adequate for pre-processing the monitoring information (feature extraction) and the general analysis of features to detect anomalies. For this kind of processing, ZEX uses a Syntactic Pattern Recogniser. This consists of a signal processor filtering monitored data, a cluster classifier that classifies observed phenomena into predefined classes identified by a symbol and finally an automaton that reads the string of these symbols and recognises the state of the phenomenon in time series.

4.3 Operating ZEX

ZEX consists of about 100 rule-based Knowledge-Sources, containing more than 1500 rules implemented with the commercial shell RTworks and several syntactic pattern recognition-based Knowledge-sources. The last version was commissioned in 1995 and has been in production since 1996.
4.3.1 The truth

Neither DEXPERT nor ZEX are as good stories as they may seem to be. For instance ZEX will be discontinued this year. The ZEUS collaboration has decided to switch off ZEX in 1998 because the cost of the maintenance (licences and manpower) is higher than the practical benefits (the automatic recovery was not fully implemented and the operator had to do the recovery by hand). Concerning DEXPERT, it is in operation and is still giving satisfaction but other “clever” elements (handling power supply trips) not based on AI techniques have been introduced into ALEPH. In fact these are written using a procedural language, i.e. FORTRAN. This type of expertise in the area of “slow control” could have been added into DEXPERT but was not done.

The reasons for the disappointing end to these stories are somehow related to the sociology of big collaborations and are not associated to the AI techniques themselves. The HEP collaborations are big, and not everybody is convinced of the advantages that this kind of technology can bring to the success of the experiment. This is similar in some ways to the questioning of the advantages of using an OO approach for software development coming from members of HEP collaborations. The other reason is that experiments run for many years and people are not permanently attached to their developments. Information somehow evaporates and so it is not unusual that some things are reinvented during the lifetime of the experiment. Finally, there is a huge inertia to introduce new computing techniques. HEP experimentalists are reluctant to introduce new computing techniques in general unless they see clear benefits.

5. LOOKING TO THE FUTURE

We must learn from the successes and failures of these pioneering experiences on using expert system techniques to assist operators running large HEP experiments. We need to apply the lessons learned during the design and implementation of the new generation of experiments, in particular we need to focus on the LHC experiments, since the future of CERN is LHC.

The Experiment Control System\(^1\) for the general purpose experiments, ATLAS and CMS are at least one order of magnitude larger than those of the LEP experiments. That is in number of sub-systems, number of parameters, diversity of equipment, etc. Operating such experiments will certainly require the aid of expert systems. We face two especially challenging problems: interfacing the Expert System with the Experiment Control System and the knowledge acquisition and long-term maintenance.

5.1 Interfacing the Expert System

It is essential to have an architecture from the beginning that foresees the intelligent assistance of the operators in charge of running the system. The Expert System should be an integral part of the system, i.e. one of many components that constitutes it. Integrated does not necessarily mean monolithic. An analogy could be to say that a spelling checker is an integral part of any word processor, however you can run a word processor without having a spelling checker.

The Expert System needs to interact with the central Error/Alarm handler of the experiment. It also needs to have access to all current and past status and monitor information in a coherent way. It is impractical if for each part of the system and sub-system the Expert System requires a different type of interface. And finally, it needs to have access to the configuration database for all the system. The proposed architecture for the Experiment Control Systems for the LHC experiments is shown in Figure 9. It foresees to have an Expert System component at the same level as other components of the system like the Alarm Handler, Data archiver, etc.

---

\(^1\) The term Experiment Control System refers to a control system that controls and monitors everything in the experiment. It includes the detector control or slow control, run control, physics data monitoring control, etc.
It is essential for interfacing the Expert System that all the functions and components of the system provide an Application Programming Interface (API). For example, it is no good if the only way to perform a given function is by means of using the mouse. If that was the case the Expert System would never be able to emulate an operator.

![Diagram of Experiment Control System architecture for the LHC experiments](image)

**Figure 9** Proposed Experiment Control System architecture for the LHC experiments

### 5.2 Knowledge Acquisition and Long-term Maintenance

Knowledge acquisition is the really challenging problem. It is inherently difficult to introduce knowledge into an Expert System. On top of that, experiments are not static; they are in continuous evolution, therefore the knowledge that is useful for the way you operate the experiment at a given moment may not work when the experiment is changed or upgraded. Only “real experts” can introduce their expertise, but not all the experts are ready to dive in with a complicated language (the case of DEXPERT) or interface.

Self-learning techniques are very attractive. The Expert System could observe the DAQ and Control system and at the same time it could spy what actions the operator is performing to solve problems and deduce the rules (learning). This technique has not been proven in our environment and it may happen that the effort needed to implement such a schema may not justify the potential benefits.

Sophisticated login of actions and errors is essential. Knowing and classifying the different problems which occur while running the experiment offers a good tool for improving the system. It is fundamental to know which problems cause the most inefficiency, and thus need to be worked on if one wants to improve the efficiency.

The life-time of the LHC experiments will be more than 10 years. The Expert System should be able to cope with changes and upgrades of the system. People other than the developers will run the experiment. It is clear that there is no silver-bullet solution for this kind of problem, however several precautions can be taken from the beginning. For example, designing the overall system with an Expert System as an integral part (later add-ons are very often problematic), providing tools that
make the introduction and changes of knowledge simple, and finally, allocating manpower for the evolution of the knowledge base.

6. CONCLUDING REMARKS

The ALEPH and ZEUS experiences are positive in demonstrating the benefits of applying AI techniques, in particular Expert Systems, for the operation of HEP experiments. The scope and the aims were a bit different for both systems. And if one look at them with some perspective, the conclusion could be that aiming at something simpler that produces clear benefits has better chances of long-term success than something sophisticated.

For the LHC experiments, Expert Systems to assist the operator running the experiment are a must. Therefore, the DAQ and Control systems need to be designed with intelligent and automated assistance in mind from the beginning.

ACKNOWLEDGEMENTS

Many people from the ALEPH and ZEUS collaborations have participated on the development and maintenance of both DAQ systems and Expert Systems. The material of this course is essentially based on their work.

BIBLIOGRAPHY

THE ARCHITECTURE OF THE HIGH PERFORMANCE STORAGE SYSTEM (HPSS)

Danny Teaff  
IBM Federal, Houston, USA

Dick Watson  
Lawrence Livermore National Laboratory, Livermore, USA

Bob Coyne  
IBM Federal, Houston, USA

Abstract
The rapid growth in the size of datasets has caused a serious imbalance in I/O and storage system performance and functionality relative to application requirements and the capabilities of other system components. The High Performance Storage System (HPSS) is a scalable, next-generation storage system that will meet the functionality and performance requirements of large-scale scientific and commercial computing environments. Our goal is to improve the performance and capacity of storage systems by two orders of magnitude or more over what is available in the general or mass marketplace today. We are also providing corresponding improvements in architecture and functionality. This paper describes the architecture and functionality of HPSS.

1. INTRODUCTION
The rapid improvement in computational science, processing capability, main memory sizes, data collection devices, multimedia capabilities, and integration of enterprise data are producing very large datasets. These datasets range from tens to hundreds of gigabytes up to terabytes. In the near future, storage systems must manage total capacities, both distributed and at single sites, scalable into the petabyte range. We expect these large datasets and capacities to be common in high-performance and large-scale national information infrastructure scientific and commercial environments. The result of this rapid growth of data is a serious imbalance in I/O and storage system performance and functionality relative to application requirements and the capabilities of other system components.

To deal with these issues, the performance and capacity of large-scale storage systems must be improved by two orders of magnitude over what is available in the general or mass marketplace today, with corresponding improvements in architecture and functionality. The goal of the HPSS collaboration is to provide such improvement. HPSS is the major development project within the National Storage Laboratory (NSL). The NSL was established to investigate, demonstrate, and commercialize new mass storage system architecture to meet the needs above [5,7,21]. The NSL and closely related projects involve more than 20 participating organization from industry, Department of Energy (DOE) and other federal laboratories, universities, and National Science Foundation (NSF) supercomputer centers. The current HPSS development team consists of IBM U.S. Federal, four DOE laboratories (Lawrence Livermore, Los Alamos, Oak Ridge, and Sandia), Cornell University, and NASA Langley and Lewis Research Centers. Ampex, IBM, Maximum Strategy Inc., Network Systems Corp., PsiTech, Sony Precision Graphics, Storage Technology, and Zitel have supplied
hardware in support of HPSS development and demonstration. Cray Research, Intel, IBM, and Meiko are cooperating in the development of high-performance access for supercomputers and MPP clients.

The HPSS commercialization plan includes availability and support by IBM as a high-end Service offering through IBM U.S. Federal. HPSS source code can also be licensed and marketed by any US. company.

Figure 1 - Example of the type of configuration HPSS is designed to support

2. ARCHITECTURAL OVERVIEW

The HPSS architecture is based on the IEEE Mass Storage Reference Model: version 5 [6,9] and is network-centered, including a high speed network for data transfer and a separate network for control (Figure 1) [4,7,13,16]. The control network uses the Open Software Foundation's (OSF) Distributed
Computing Environment DCE Remote Procedure Call technology [17]. In actual implementation, the control and data transfer networks may be physically separate or shared. An important feature of HPSS is its support for both parallel and sequential input/output (I/O) and standard interfaces for communication between processors (parallel or otherwise) and storage devices. In typical use, clients direct a request for data to an HPSS server. The HPSS server directs the network-attached storage devices or servers to transfer data directly, sequentially or in parallel to the client node(s) through the high speed data transfer network. TCP/IP sockets and IPI-3 over High Performance Parallel Interface (HIPPI) are being utilized today; Fibre Channel Standard (FCS) with IPI-3 or SCSI, or Asynchronous Transfer Mode (ATM) will also be supported in the future [3,20,22]. Through its parallel storage support by data striping HPSS will continue to scale upward as additional storage devices and controllers are added to a site installation.

The key objectives of HPSS are now described.

2.1 Scalability

A major driver for HPSS is to develop a scalable, distributed, high performance storage management system. HPSS is designed to scale in several dimensions.

The HPSS I/O architecture is designed to provide I/O performance scaling by supporting parallel I/O through software striping [1]. The system will support application data transfers from megabytes to gigabytes per second with total system throughput of many gigabytes per second. Data object number and size must scale to support billions of data objects, each potentially terabytes or larger in size, for total storage capacities in petabytes. This is accomplished through 64-bit metadata fields and scalable organization of system metadata. The system also is required to scale geometrically to support distributed systems with hierarchies of hierarchical storage systems.

Multiple storage systems located in different areas must integrate into a single logical system accessible by personal computers, workstations, and supercomputers. These requirements are accomplished using a client/server architecture, the use of OSF's DCE as its distributed infrastructure, support for distributed file system interfaces and multiple servers. HPSS also supports a scalable storage object name service capable of managing millions of directories and the ability to support hundreds to thousands of simultaneous clients. The latter is achieved through the ability to multitask, multiprocess and replicate the HPSS servers.

2.2 Modularity and APIs

The HPSS architecture is highly modular. Each replicable software component is responsible for a set of storage objects, and acts as a service provider for those objects. The IEEE Reference Model, on which the HPSS design is based, provides the modular layered functionality (see Figure 2) [6,9]. The HPSS software components are loosely coupled, with open application program interfaces (APIs) defined at each component level. Most users will access HPSS at its high level interfaces—currently client API, FTP (both parallel and sequential), NFS, Parallel File System (PFS), with AFS/DFS, Unix Virtual File System (VFS), and Data Management Interface Group (DMIG) interfaces in the future) [11,15,18,19]. However, APIs are available to the underlying software components for applications, such as large scale data management, digital library or video-on-demand requiring high performance or special services. This layered architecture affords the following advantages:

- Replacement of selected software components—As new and better commercial software and hardware components became available, an installation can add or replace existing components. For example, an installation might add or replace Physical Volume Repositories, Movers or the HPSS Physical Volume Library with other commercially available products.

- Support of applications direct access to lower level services—The layered architecture is designed to accommodate efficient integration of different applications such as digital library, object store, multimedia, and data management systems. Its modularity will enable HPSS to be embedded transparently into the large distributed information management systems that will form the
information services in the emerging national information infrastructure. Support for different name spaces or data organizations is enabled through introduction of new Name Servers and data management applications.

2.3 Portability and Standards

Another important design goal is portability to many vendors' platforms to enable OEM and multivendor support of HPSS. HPSS has been designed to run under Unix requiring no kernel modifications, and to use standards based protocols, interfaces, and services where applicable. HPSS is written in ANSI C, and uses POSIX functions to enhance software portability. Use of existing commercial products for many of the infrastructure services supported on multiple-vendor platforms enables portability, while also providing market proven dependability. Open Software Foundation (OSF) Distributed Computing Environment (DCE), Transarc's Encina transaction manager [8], Kinesix SAMMI and X-windows are being used by HPSS because of their support across multiple vendor platforms, in addition to the rich set of functionality provided. The HPSS component APIs have been turned over to the IEEE Storage System Standards Working Group as a basis for its standards activities.

2.4 Reliability and Recovery

Reliable and recoverable storage of data is mandatory for any storage system. HPSS supports several mechanisms to facilitate this goal. The client-server interactions between HPSS software components have been designed to be based on atomic transactions in order to maintain system state consistency [14]. Within the scope of a given request, a transaction may be established so that an abort (or commit) in one component will cause the other participating components to abort (or commit). The HPSS Metadata Manager is fully integrated with its Transaction Manager. Following an abort, the non-volatile file and name space metadata changes within the scope of the transactions will automatically be rolled back. For recovery purposes, mirroring of the storage object and name space metadata is supported. The HPSS architecture will also support data mirroring if desired in a future release.

Support is also provided to recover from failed devices and bad media. An administrator interface is provided to place a device off line. Once the device has been repaired, it may then be placed back on line. For bad media, an application interface is provided to move storage segments from a virtual volume to a new virtual volume.

The HPSS software components execute in a distributed manner. Should a processor fail, any of the HPSS software components may be moved to another platform. Component services are registered with the DCE Cell Directory Service (CDS) so that components may locate the services. Each component has also been designed to perform reconnect logic when a connection to a peer component fails. Connection context is maintained by selected components. When a connection context is established, a keep-alive activity is started to detect broken connections. A server may use the context information associated with a broken connection to perform any necessary clean up.

2.5 Security and Privacy

HPSS uses DCE and POSIX security and privacy mechanisms for authentication, access control lists, permissions and security labels. Security policy is handled by a separate policy module. Audit trails are also supported. Further, HPSS design and implementation use a rigorous software engineering methodology which support its reliability and maintainability.

2.6 Storage System Management

HPSS has a rich set of storage system management services for operators and system administrators based on managed object definitions. The application programming interface supports monitoring, reporting and controlling operations (see Appendix A).
3. SOFTWARE COMPONENTS

The HPSS software components are shown Figure 2. The shaded boxes are defined in the IEEE Mass Storage Reference Model: version 5 [9].

![Software Model Diagram](image)

Figure 2 - Software Model Diagram

This section outlines the function of each component.

3.1 Infrastructure

HPSS design is based upon a well-formed industry standard infrastructure. The key infrastructure components are now outlined.

3.1.1 Distributed Computing Environment

HPSS uses OSF's DCE as the base infrastructure for its distributed architecture [17]. This standards-based framework will enable the creation of distributed storage systems for a national information infrastructure capable of handling gigabyte-terabyte-class files at gigabyte per second data transfer rates.
DCE was selected because of its wide adoption among vendors and its near industry-standard status. HPSS uses the DCE Remote Procedure Call (RPC) mechanism for control messages and DCE Threads for multitasking. The DCE threads package is vital for HPSS to serve large numbers of concurrent users and to enable multiprocessing of its servers. HPSS also uses the DCE Security, Cell Directory, and Time services. A library of DCE convenience functions was developed for use in HPSS.

### 3.1.2 Transaction Management

Requests to HPSS to perform actions such as creating_bitfiles or accessing file data results in client/server interactions between software components. Transaction integrity is required to guarantee consistency of server state and metadata in case a particular component should fail. As a result, a transaction manager was required by HPSS. Encina, from Transarc, was selected by the HPSS project as its transaction manager [8]. This selection was based on functionality, its use of DCE, and multi-platform vendor support.

Encina provides begin-commit-abort semantics, distributed two-phase commit, and nested transactions. In addition, Transaction RPCs (TRPCs), which extend DCE RPCs with transaction semantics, are provided. For recovery purposes, Encina uses a write-ahead log for storing transaction outcomes and updates to recoverable metadata. Mirroring of data is also provided.

```c
infunctor();
  transaction {
    ...
  }
onCommit
    ...
onAbort
    ...
code structure
```

### 3.1.3 Metadata Management

Each HPSS software component has system metadata associated with the objects it manages. Each server with non-volatile metadata requires the ability to reliably store its metadata. It is also required
that metadata management performance be scalable as the number of object instances grow. In addition, access to metadata by primary and secondary keys is required. The Structured File Server (SFS), an Encina optional product, was selected by the HPSS project as its metadata manager. SFS provides B-tree clustered file records, record and field level access, primary and secondary keys, and automatic byte ordering between machines. SFS is also fully integrated with the Encina transaction manager. As a result, SFS provides transaction consistency and data recovery from transaction aborts. For reliability purposes, HPSS metadata stored in SFS is mirrored. A library of metadata manager convenience functions for retrieving, adding, updating, and deleting metadata for each of the HPSS components was developed.

![Figure 5 - Structured File Server (SFS)](image)

### 3.1.4 Security

The security components of HPSS provide authentication, authorization, enforcement, and audit capabilities for the HPSS components. Authentication is responsible for guaranteeing that a principal is the entity that is claimed, and that information received from an entity is from that entity. Authorization is responsible for enabling an authenticated entity access to an allowed set of resources and objects. Authorization enables end user access to HPSS directories and bitfiles. Enforcement is responsible for guaranteeing that operations are restricted to the authorized set of operations. Enforcement applies to end user access to bitfiles. Audit is responsible for generating a log of security relevant activity. HPSS security libraries utilize DCE and DCE security. The authentication service, which is part of DCE, is based on Kerberos v5. The following figure depicts how HPSS security fits with DCE and Kerberos.
3.1.5 Communication

The control path communications between HPSS components is through DCE RPCs or Encina transaction RPCs. For data path communication, the HPSS Mover(s) currently utilize either Sockets or IPI-3 (over HIPPI) libraries. Future support is planned for IPI-3 and SCSI over Fibre Channel Standard and TCP/IP over ATM. A special parallel data transfer library has been developed. This library allows data to be transferred across many parallel data connections. The library transfers data headers that identify the data that follows. This allows data to be sent and arrive in any order on the parallel paths.

3.1.6 Logging

The HPSS logger is used to record alarms, events, requests, security audit records, accounting records, and trace information from the HPSS components. A central log is maintained which contains records from all HPSS components. A local log of activity from components on each HPSS node is also supported. When the central log fills, it will switch to a secondary log file. A configuration option allows the filled log to be automatically archived to HPSS. A delog function is provided to extract and format log records. Delog options support filtering by time interval, record type, server, and user.

3.1.7 64 Bit Arithmetic Libraries

HPSS supports file sizes up to \(2^{**64}\) bytes. Many vendor platforms support only 32 bit integer arithmetic. In order to support large file sizes and large numbers of objects on 32 bit platforms, a library of 64 bit arithmetic functions has been developed. The functions support both big endian and little endian I/O architectures.

3.2 Interfaces

HPSS supports several high-level interfaces: currently Client API, FTP (both standard and parallel), and NFS, with DFS/AFS, DMIG, and VFS planned for future releases.

3.2.1 Client API

The HPSS Client file server API mirrors the POSIX file system interface specification where possible. The Client API also supports extensions to allow the programmer to take advantage of the
specific features provided by HPSS (e.g., class-of-service, storage/access hints passed at file creation and support for parallel data transfers).

3.2.2 FTP (standard and parallel)

HPSS provides a standard FTP server interface to transfer files from HPSS to a local file system. Parallel FTP, an extension and superset of standard FTP, has been implemented to provide high performance data transfers to client systems. The standard FTP protocol supports third-party data transfer through separation of the data transfer and control paths, but it does not offer parallel data paths [11]. HPSS modified and augmented the standard client FTP file retrieval and storage functions to offer parallel data paths for HPSS data transfers. This approach provides high performance FTP transfers to the client while still supporting the FTP command set. Additional commands have been added to support parallel transfer. This work will be submitted to the Internet Engineering Task Force for standardization.

3.2.3 NFS

The NFS V2 Server interface for HPSS provides transparent access to HPSS name space objects and bitfile data for client systems from both the native HPSS and the Network File System V2 service. The NFS V2 Server translates standard NFS calls into HPSS control calls and provides data transfers for NFS read and write requests. The NFS V2 Server handles optimization of data movement requests by the caching of data and control information. If the server machine crashes, the NFS V2 Server is in charge of recovery of all cached data at the time of the crash. The NFS V2 Server will also recover when HPSS crashes. Before NFS clients can request NFS services, they must mount an exported HPSS directory by calling the Mount daemon mount API. Support for NFS V3 is planned for a future release.

3.2.4 Parallel File System

HPSS provides the capability to act as an external hierarchical file system to vendor Parallel File Systems (PFS). The first implementation supports the IBM SPx PIOFS. Early deployment is also planned for Intel Paragon and Meiko PFS integration with HPSS.

3.3 Name Server (NS)

The Name Server maps a file name to an HPSS object. The Name Server provides a POSIX view of the name space which is a hierarchical structure consisting of directories, files, and links. File names are human readable ASCII strings. Namable objects are any object identified by HPSS Storage Object IDs. The commonly named objects are bitfiles, directories, or links. In addition to mapping names to unique object identifiers, the Name Server provides access verification to objects. POSIX Access Control Lists (ACLs) are supported for the name space objects. A key requirement of the Name Server is to be able to scale to millions of directories and greater than a billion name space entries.

3.4 Bitfile Server (BFS)

The Bitfile Server provides the POSIX file abstraction to its clients. A logical bitfile is an uninterpreted bit string. HPSS supports bitfile sizes up to 2**64 bytes. A bitfile is identified by a Bitfile Server generated name called a bitfile-id. Mapping of a human readable name to the bitfile id is provided by a Name Server external to the Bitfile Server. Clients may reference portions of a bitfile by specifying the bitfile-id and a starting address and length. The writes and reads to a bitfile are random and the writes may leave holes where no data has been written. The Bitfile Server supports both sequential and parallel read and write of data to bitfiles. In conjunction with Storage Servers, the Bitfile Server maps logical portions of bitfiles onto physical storage devices.

3.5 Storage Server (SS)

The Storage Server provides a hierarchy of storage objects: logical storage segments, virtual volumes and physical volumes. All three layers of the Storage Server can be accessed by appropriately
privileged clients. The server translates references to storage segments into references to virtual volume and finally into physical volume references. It also schedules the mounting and dismounting of removable media through the Physical Volume Library. The Storage Server in conjunction with the Mover have the main responsibility for orchestration of HPSS's parallel I/O operations.

The storage segment service is the conventional method for obtaining and accessing HPSS storage resources. The Storage Server maps an abstract storage space, the storage segment, onto a virtual volume, resolving segment addresses as required. The client is presented with a storage segment address space, with addresses from 0 to N-1, where N is the byte length of the segment. Segments can be opened, created, read, written, closed and deleted. Characteristics and information about segments can be retrieved and changed.

The virtual volume service is the method provided by the Storage Server to group physical storage volumes. The virtual volume service supports striped volumes today and mirrored volume in a future release. Thus, a virtual volume can span multiple physical volumes. The Storage Server maps the virtual volume address space onto the component physical volumes in a fashion appropriate to the grouping. The client is presented with a virtual volume that can be addressed from 0 to N-1, where N is the byte length of the virtual volume. Virtual volumes can be mounted, created, read, written, unmounted and deleted. Characteristics of the volume can be retrieved and in some cases, changed.

The physical volume service is the method provided by the storage server to access the physical storage volumes in HPSS. Physical volumes can be mounted, created, read, written, unmounted and deleted. Characteristics of the volume can be retrieved and in some cases, changed.

Repack runs as a separate process. It provides defragmentation of physical volumes. Repack utilizes a Storage Server provided function which moves storage segments to a different virtual volume.

### 3.6 Mover (Mvr)

The Mover is responsible for transferring data from a source device(s) to a sink device(s). A device can be a standard I/O device with geometry (e.g., a tape or disk), or a device without geometry (e.g., network, memory). The Mover also performs a set of device control operations. Movers perform the control and transfer of both sequential and parallel data transfers.

The Mover consists of several major parts: Mover parent task, Mover listen task/request processing task, Data Movement, Device control, and System Management.

The Mover parent task performs Mover initialization functions, and spawns processes to handle the Mover's DCE communication, data transfer connections, as well as the Mover's functional interface. The Mover listen task listens on a well-known TCP port for incoming connections to the Mover, spawns request processing tasks, and monitors completion of those tasks. The request processing task performs initialization and return functions common to all Mover requests. Data movement supports client requests to transfer data to or from HPSS. Device control supports querying the current device read/write position, changing the current device read/write position, loading a physical volume into a drive, unloading a physical volume from a drive, flushing data to the media, writing a tape mark, loading a message to a device's display area, reading a media label, writing a media label, and zeroing a portion of disk. System management supports querying and altering device characteristics and overall Mover state.

### 3.7 Physical Volume Library (PVL)

The PVL manages all HPSS physical volumes. Clients can ask the PVL to mount and dismount sets of physical volumes. Clients can also query the status and characteristics of physical volumes. The PVL maintains a mapping of physical volume to cartridge and a mapping of cartridge to PVR. The PVL also controls all allocation of drives. When the PVL accepts client requests for volume mounts, the PVL allocates resources to satisfy the request. When all resources are available, the PVL issues
commands to the PVR(s) to mount cartridges in drives. The client is notified when the mount has completed.

The Physical Volume Library consists of two major parts: Volume mount service and Storage system management service.

The volume mount service is provided to clients such as a Storage Server. Multiple physical volumes belonging to a virtual volume may be specified as part of a single request. All of the volumes will be mounted before the request is satisfied. All volume mount requests from all clients are handled by the PVL. This allows the PVL to prevent multiple clients from deadlocking when trying to mount intersecting sets of volumes. The standard mount interface is asynchronous. A notification is provided to the client when the entire set of volumes has been mounted. A synchronous mount interface is also provided. The synchronous interface can only be used to mount a single volume, not sets of volumes. The synchronous interface might be used by a non-HPSS process to mount cartridges which are in a tape library, but not part of the HPSS system.

The storage system management service is provided to allow a management client control over HPSS tape repositories. Interfaces are provided to import, export, and move volumes. When volumes are imported into HPSS, the PVL is responsible for writing a label to the volume. This label can be used to confirm the identity of the volume every time it is mounted. Management interfaces are also provided to query and set the status of all hardware managed by the PVL (volumes, drives, and repositories).

### 3.8 Physical Volume Repository (PVR)

The PVR manages all HPSS supported robotics devices and their media such as cartridges. Clients can ask the PVR to mount and dismount cartridges. Every cartridge in HPSS must be managed by exactly one PVR. Clients can also query the status and characteristics of cartridges.

The Physical Volume Repository consists of these major parts: Generic PVR service, and support for devices such as Ampex, STK, and 3494/3495 robot services, as well as an operator mounted device service.

The generic PVR service provides a common set of APIs to the client regardless of the type of robotic device being managed. Functions to mount, dismount, inject and eject cartridges are provided. Additional functions to query and set cartridge metadata are provided. The mount function is asynchronous. The PVR calls a well-known API in the client when the mount has completed. For certain devices, like operator mounted repositories, the PVR will not know when the mount has completed. In this case it is up to the client to determine when the mount has completed. The client may poll the devices or use some other method. When the client determines a mount has completed, the client should notify the PVR using one of the PVR's APIs. All other PVR functions are synchronous. The generic PVR maintains metadata for each cartridge managed by the PVR. The generic PVR interface calls robotics vendor supplied code to manage specific robotic devices.

The operator mounted device service manages a set of cartridges that are not under the control of a robotics device. These cartridges are mounted to a set of drives by operators. The Storage System Manager is used to inform the operators when mount operations are required.

### 3.9 Storage System Management (SSM)

The HPSS SSM architecture is based on the ISO managed object architecture [10,12]. The Storage System Manager (SSM) monitors and controls the available resources of the HPSS storage system in ways that conform to the particular management policies of a given site. Monitoring capabilities include the ability to query the values of important management attributes of storage system resources as well as an ability to receive notifications of alarms and other significant system events. Controlling capabilities include the ability to set the values of management attributes of storage system resources and storage system policy parameters. Additionally, SSM can request that specific
operations be performed on resources within the storage system, such as adding and deleting logical or physical resources. The operations performed by SSM are usually accomplished through standard HPSS server APIs.

SSM management roles cover a wide spectrum, including configuration aspects of installation, creating new volumes, initialization, operations, and termination tasks. SSM can provide management capabilities to a range of clients, including site administrators, systems administrators, operations personnel, complex graphical user interface (GUI) management environments, and independent management applications responsible for tasks such as purges, migration, and reclamation. Some of the functional areas of SSM include fault management, configuration management, security management, accounting management, and performance management.

SSM consists of these major parts: SSM Graphical User Interface (SAMMI GUI Displays), SAMMI Data Server, and System Manager.

The SSM Graphical User Interface allows operators, administrators, and users to interactively monitor and control the HPSS storage system. Kinesix's SAMMI product is used to provide the HPSS GUI services. SAMMI is built on X-windows and OSF's Motif. It provides mechanisms to simplify screen design and data management services for screen fields. Standard Motif widgets such as menus, scrollbar lists, and buttons are used. In addition SAMMI specific widgets such as dials, gauges, and bar charts are used for informational and statistical data.

The SAMMI Data Server is a client to the System Manager and a server to the SAMMI Runtime Display Manager. The SAMMI Data Server is the means by which data is acquired and fed to the SAMMI Displays.

The Storage System Manager is a client to the HPSS servers and a server to the SAMMI Data Server and other external clients wishing to perform management specific operations. It interfaces to the managed objects defined by the HPSS servers.

**Figure 7 - Storage System Management**

### 3.9.1 Migration - Purge

The Migration-Purge server provides hierarchical storage management for HPSS through migration and caching of data between devices. There are two types of migration and caching: disk migration and caching and tape migration and caching. Multiple storage hierarchies are supported by HPSS [2]. Data is cached to the highest level (fastest) device in a given hierarchy when accessed and migrated when inactive and space is required.
The main purpose of disk migration is to free up the disk storage. This type of migration contains two functions; migration and purge. Migration selects the qualified bitfiles and copies these bitfiles to the next storage level defined in the hierarchy. Purge later frees the original bitfiles from the disk storage.

The main purpose of tape migration is to free up tape volumes, and not just migrate bitfiles. The active bitfiles in the target virtual volumes are moved laterally to the free tape volumes in the same storage level. The inactive bitfiles in the target virtual volumes are migrated to the free tape volumes in the next storage level.

The HPSS component client APIs provide the vehicle for the Storage System Manager to request the server to start migration and purge whenever it is necessary. The migration-purge server is set up to run migration periodically with the time interval specified in the migration policy. In addition, the server will start the migration and purge to run automatically if the free space of a storage class is below the percentage specified in the migration-purge policy.

4. OTHER

4.1 Installation

Installation software is provided for system administrators to install/update HPSS, and perform the initial configuration of HPSS following installation. The full HPSS system is first installed to an installation node. Selected HPSS software components may then be installed (using the remote installation feature) from the installation node to the other nodes where HPSS components will be executed.

4.2 NSL-UniT Migration

HPSS, through its support of parallel storage, provides significant improvements in I/O rates and storage capacity over existing storage systems software. In transitioning from existing systems, a migration path is required. The migration path should be transparent to end users of the storage system. The capability to migrate from NSL UniTree to HPSS is provided. The migration software handles both file metadata and actual data. Utilities convert the file metadata (e.g., storage maps, virtual volume data, physical volume data), and name space metadata from UniTree format to HPSS format. Actual data is not moved. The HPSS Mover software contains additional read logic to recognize NSL UniTree data formats when an NSL UniTree file is accessed. Utilities to support migration from other legacy storage systems will also be provided as required.

4.3 Accounting

HPSS provides interfaces to collect accounting information (initially storage space utilization). These interfaces may be used by site specific programs to charge for data storage. SSM provides user interfaces to run the accounting collection utility, change account numbers and change the account code assigned to storage objects.

5. SUMMARY AND STATUS

We have described the key objectives, features and components of the HPSS architecture. At the time this paper is being written, December 1994, HPSS Release 1 (R1) is in integration testing and planning for its early deployment at several sites has begun. R1 contains all the basic HPSS components and services and supports parallel tape. It is targeted at MPP environments with existing parallel disk services. Much of the coding for Release 2 (R2) has been completed also. R2 adds support for parallel disks, migration and caching between levels of the hierarchy and other functionality. R2 will be a complete stand-alone system and is targeted for third quarter 1995.

We demonstrated, HPSS at Supercomputing 1994 with R1 and early R2 capabilities of parallel disks, and tape access (Ampex D2, IBM NTP and 3490), to an IBM SP2, IBM RS 6000, PsiTech
framebuffer, and Sony high-resolution monitor over a NSC HIPPI switch. HPSS R1 is on order 95K lines of executable source code and R2 is expected to add on another 50K lines of executable source code.

Our experience indicates that the architectural choices of basing the system on the IEEE Reference Model, use of an industry defacto standard infrastructure based on OSF DCE and Transarc Encina, and use of other industry standards such as POSIX, C, Unix, ISO managed object model for Storage System Management and standard communication protocols is sound. This foundation plus the software engineering methodology employed, we believe, positions HPSS for a long and useful life for both scientific and commercial high performance environments.

6. ACKNOWLEDGMENTS

We wish to acknowledge the many discussions and shared design, implementation, and operation experiences with our colleagues in the National Storage Laboratory collaboration, the IEEE Mass Storage Systems and Technology Technical Committee, the IEEE Storage System Standards Working Group, and in the storage community. Specifically we wish to acknowledge the people on the HPSS Technical Committee and Development Teams. At the risk of leaving out a key colleague in this ever-growing collaboration, the authors wish to acknowledge Dwight Barrus, Ling-Ling Chen, Ron Christman, Danny Cook, Lynn Kluegel, Tyce McLarty, Christina Mercier, and Bart Parliman from LANL; Larry Berdahl, Jim Daveler, Dave Fisher, Mark Gary, Steve Louis, Donna Mecozzi, Jim Minton, and Norm Samuelson from LLNL; Marty Barnaby, Rena Haynes, Hilary Jones, Sue Kelly, and Bill Rahe from SNL; Randy Burris, Dan Million, Daryl Steiniert, Vicky White, and John Wingenbach from ORNL; Donald Creig Humes, Juliet Pao, Travis Priest and Tim Starrin from NASA LaRC; Andy Hanushevsky, Lenny Silver, and Andrew Wyatt from Cornell; and Paul Chang, Jeff Deutsch, Kurt Everson, Rich Ruef, Tracy Tran, Terry Tyler, and Benny Wilbanks from IBM U.S. Federal and its contractors.

This work was, in part, performed by the Lawrence Livermore National Laboratory, Los Alamos National Laboratory, Oak Ridge National Laboratory, and Sandia National Laboratories, under auspices of the U.S. Department of Energy Cooperative Research and Development Agreements, by Cornell, Lewis Research Center and Langley Research Center under auspices of the National Aeronautics and Space Agency and by IBM U.S. Federal under Independent Research and Development and other internal funding.

7. REFERENCES


11. Internet Standards. The official Internet standards are defined by RFC's (TCP protocol suite). RFC 783; TCP standard defined. RFC 959; FTP protocol standard. RFC 1068; FTP use in third-party transfers. RFC 1094; NFS standard defined. RFC 1057; RPC standard defined.


APPENDIX A

Application Programming Interfaces (APIs) to HPSS Components

HPSS provides an application client library containing file, directory, and client state operations.

The HPSS Client Library provides the following routines grouped by related functionality.

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpss_Open</td>
<td>client</td>
<td>Optionally create and open an HPSS file</td>
</tr>
<tr>
<td>hpss_Close</td>
<td>client</td>
<td>Close a file</td>
</tr>
<tr>
<td>hpss_Umask</td>
<td>client</td>
<td>Set the file creation mask</td>
</tr>
<tr>
<td>hpss_Read</td>
<td>client</td>
<td>Read a contiguous section of an HPSS file, beginning at the current file offset</td>
</tr>
<tr>
<td>hpss_Write</td>
<td>client</td>
<td>Write data from a client buffer to a contiguous section of an HPSS file</td>
</tr>
<tr>
<td>hpss_Lseek</td>
<td>client</td>
<td>Reposition the read/write file offset</td>
</tr>
<tr>
<td>hpss_ReadList</td>
<td>client</td>
<td>Read data from an HPSS file, specifying lists for data sources and sinks</td>
</tr>
<tr>
<td>hpss_WriteList</td>
<td>client</td>
<td>Write data to an HPSS file, specifying lists for data sources and sinks</td>
</tr>
<tr>
<td>hpss_Sstat</td>
<td>client</td>
<td>Get file status</td>
</tr>
<tr>
<td>hpss_Fstat</td>
<td>client</td>
<td>Get file status</td>
</tr>
<tr>
<td>hpss_lstat</td>
<td>client</td>
<td>Get file status, returning status about a symbolic link if the named file is a symbolic link</td>
</tr>
<tr>
<td>hpss_FileGetAttributes</td>
<td>client</td>
<td>Get attributes for a file</td>
</tr>
<tr>
<td>hpss_FileSetAttributes</td>
<td>client</td>
<td>Alter file attribute values</td>
</tr>
<tr>
<td>hpss_Access</td>
<td>client</td>
<td>Check file accessibility</td>
</tr>
<tr>
<td>hpss_Chmod</td>
<td>client</td>
<td>Change the file mode of an HPSS file</td>
</tr>
<tr>
<td>hpss_Chown</td>
<td>client</td>
<td>Change owner and group of an HPSS file</td>
</tr>
<tr>
<td>hpss_Utime</td>
<td>client</td>
<td>Set access and modification times of an HPSS file</td>
</tr>
<tr>
<td>hpss_GetACL</td>
<td>client</td>
<td>Query the Access Control List of a file</td>
</tr>
<tr>
<td>hpss_DeleteACLEntry</td>
<td>client</td>
<td>Remove an entry from the Access Control List of a file</td>
</tr>
<tr>
<td>hpss_UpdateACLEntry</td>
<td>client</td>
<td>Update an entry in the Access Control List of a file</td>
</tr>
<tr>
<td>hpss_Truncate</td>
<td>client</td>
<td>Set the length of a file</td>
</tr>
<tr>
<td>hpss_Ftruncate</td>
<td>client</td>
<td>Set the length of a file</td>
</tr>
<tr>
<td>hpss_Fclear</td>
<td>client</td>
<td>Clear part of a file</td>
</tr>
<tr>
<td>hpss_Cache</td>
<td>client</td>
<td>Cache a piece of a file to a specified level in the storage hierarchy</td>
</tr>
<tr>
<td>hpss_Fcache</td>
<td>client</td>
<td>Cache a piece of a file to a specified level in the storage hierarchy</td>
</tr>
<tr>
<td>hpss_Purge</td>
<td>client</td>
<td>Purge a piece of a file from a specified level in the storage hierarchy</td>
</tr>
<tr>
<td>hpss_Fpurge</td>
<td>client</td>
<td>Purge a piece of a file from a specified level in the storage hierarchy</td>
</tr>
<tr>
<td>hpss_Migrate</td>
<td>client</td>
<td>Migrate a piece of a file from a specified level in the storage hierarchy</td>
</tr>
<tr>
<td>hpss_Fmigrate</td>
<td>client</td>
<td>Migrate a piece of a file from a specified level in the storage hierarchy</td>
</tr>
<tr>
<td>hpss_Link</td>
<td>client</td>
<td>Create a hard link to an existing HPSS file</td>
</tr>
<tr>
<td>hpss_Unlink</td>
<td>client</td>
<td>Remove an entry from an HPSS directory</td>
</tr>
<tr>
<td>hpss_Rename</td>
<td>client</td>
<td>Rename a file or directory</td>
</tr>
<tr>
<td>hpss_Symlink</td>
<td>client</td>
<td>Create a symbolic link</td>
</tr>
<tr>
<td>hpss_Readlink</td>
<td>client</td>
<td>Read the contents of a symbolic link (i.e., the data stored in the symbolic link)</td>
</tr>
<tr>
<td>hpss_Mkdir</td>
<td>client</td>
<td>Create a directory</td>
</tr>
<tr>
<td>hpss_Rmdir</td>
<td>client</td>
<td>Remove an HPSS directory</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>hpss_Opendir</td>
<td>Open an HPSS directory</td>
<td></td>
</tr>
<tr>
<td>hpss_Readdir</td>
<td>Read a directory entry</td>
<td></td>
</tr>
<tr>
<td>hpss_Rewinddir</td>
<td>Reset position of an open directory stream</td>
<td></td>
</tr>
<tr>
<td>hpss_Closedir</td>
<td>Close an open directory stream</td>
<td></td>
</tr>
<tr>
<td>hpss_Chdir</td>
<td>Change current working directory</td>
<td></td>
</tr>
<tr>
<td>hpss_Getcwd</td>
<td>Get current working directory</td>
<td></td>
</tr>
<tr>
<td>hpss_Chroot</td>
<td>Change the root directory for the current client</td>
<td></td>
</tr>
<tr>
<td>hpss_LoadThreadState</td>
<td>Updates the user credentials and file/directory creation mask for a thread's API state</td>
<td></td>
</tr>
<tr>
<td>hpss_ThreadCleanup</td>
<td>Cleans up a thread's Client API state</td>
<td></td>
</tr>
<tr>
<td>hpss_AccessHandle</td>
<td>Determines client accessibility to a file, given a Name Server object handle and file pathname</td>
<td></td>
</tr>
<tr>
<td>hpss_OpenBitfile</td>
<td>Opens and HPSS file, specified by bitfile ID</td>
<td></td>
</tr>
<tr>
<td>hpss_OpenHandle</td>
<td>Open an HPSS file, specified by Name Server object ID and, optionally, pathname</td>
<td></td>
</tr>
<tr>
<td>hpss_GetAttrHandle</td>
<td>Get attributes of an HPSS file, specified by Name Server object ID and, optionally, pathname</td>
<td></td>
</tr>
<tr>
<td>hpss_SetAttrHandle</td>
<td>Set attributes of an HPSS file, specified by Name Server object ID and, optionally, pathname</td>
<td></td>
</tr>
<tr>
<td>hpss_GetACLHandle</td>
<td>Query the Access Control List of a file</td>
<td></td>
</tr>
<tr>
<td>hpss_DeleteACLEntry-Handle</td>
<td>Remove an entry from the Access Control List of a file</td>
<td></td>
</tr>
<tr>
<td>hpss_UpdateACLEntry-Handle</td>
<td>Update an entry in the Access Control List of a file</td>
<td></td>
</tr>
<tr>
<td>hpss_LinkHandle</td>
<td>Create a hard link to an existing HPSS file, given the name space object handle of the existing object, and relative directory for the new link and the pathname of the new link</td>
<td></td>
</tr>
<tr>
<td>hpss_LookupHandle</td>
<td>Query the Name Server to obtain attributes, an access ticket and object handle for a specified name space entry</td>
<td></td>
</tr>
<tr>
<td>hpss_MkdirHandle</td>
<td>Create a new directory</td>
<td></td>
</tr>
<tr>
<td>hpss_RmdirHandle</td>
<td>Remove a directory</td>
<td></td>
</tr>
<tr>
<td>hpss_ReaddirHandle</td>
<td>Read directory entries</td>
<td></td>
</tr>
<tr>
<td>hpss_UnlinkHandle</td>
<td>Remove directory entry</td>
<td></td>
</tr>
<tr>
<td>hpss_RenameHandle</td>
<td>Rename a directory entry</td>
<td></td>
</tr>
<tr>
<td>hpss_SymlinkHandle</td>
<td>Create a symbolic link</td>
<td></td>
</tr>
<tr>
<td>hpss_ReadlinkHandle</td>
<td>Read the contents of a symbolic link</td>
<td></td>
</tr>
<tr>
<td>hpss_TruncateHandle</td>
<td>Set the length of a file</td>
<td></td>
</tr>
<tr>
<td>hpss_StageHandle</td>
<td>Stage a piece of a file to a specified level in the storage hierarchy</td>
<td></td>
</tr>
<tr>
<td>hpss_PurgeHandle</td>
<td>Purge a piece of a file from a specified level in the storage hierarchy</td>
<td></td>
</tr>
<tr>
<td>hpss_MigrateHandle</td>
<td>Migrate a piece of a file from a specified level in the storage hierarchy</td>
<td></td>
</tr>
</tbody>
</table>
The Name Server provides APIs for the following operations:

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ns_Insert</td>
<td>client</td>
<td>Insert a bitfile object into a directory</td>
</tr>
<tr>
<td>ns_Delete</td>
<td>client</td>
<td>Delete a name space object</td>
</tr>
<tr>
<td>ns_Rename</td>
<td>client</td>
<td>Rename a name space object</td>
</tr>
<tr>
<td>ns_MkLink</td>
<td>client</td>
<td>Create a hard link to file</td>
</tr>
<tr>
<td>ns_MkSymLink</td>
<td>client</td>
<td>Make a symbolic link</td>
</tr>
<tr>
<td>ns_ReadLink</td>
<td>client</td>
<td>Read data associated with a symbolic link</td>
</tr>
<tr>
<td>ns_GetName</td>
<td>client</td>
<td>Get path name for the specified bitfile</td>
</tr>
<tr>
<td>ns_GetACL</td>
<td>client</td>
<td>Get an ACL for the specified name server object</td>
</tr>
<tr>
<td>ns_SetACL</td>
<td>client</td>
<td>Set an ACL for the specified name server object</td>
</tr>
<tr>
<td>ns_DeleteACEntry</td>
<td>client</td>
<td>Delete an entry from the ACL of the specified name server object</td>
</tr>
<tr>
<td>ns_UpdateACEntry</td>
<td>client</td>
<td>Update an entry from the ACL of the specified name server object</td>
</tr>
<tr>
<td>ns_Mkdir</td>
<td>client</td>
<td>Create a directory</td>
</tr>
<tr>
<td>ns_ReadDir</td>
<td>client</td>
<td>Return a list of directory entries</td>
</tr>
<tr>
<td>ns_GetAttrs</td>
<td>SSM, client</td>
<td>Get Name Server handle and managed object attributes</td>
</tr>
<tr>
<td>ns_SetAttrs</td>
<td>SSM, client</td>
<td>Set Name Server managed object attributes</td>
</tr>
</tbody>
</table>

The Bitfile Server provides APIs for the following operations:

<table>
<thead>
<tr>
<th>API</th>
<th>Client</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bfs_Create</td>
<td>client</td>
<td>Create a bitfile</td>
</tr>
<tr>
<td>bfs_Unlink</td>
<td>client</td>
<td>Unlink a bitfile</td>
</tr>
<tr>
<td>bfs_Open</td>
<td>client</td>
<td>Open a bitfile</td>
</tr>
<tr>
<td>bfs_Close</td>
<td>client</td>
<td>Close a bitfile</td>
</tr>
<tr>
<td>bfs_Read</td>
<td>client</td>
<td>Read data from a bitfile</td>
</tr>
<tr>
<td>bfs_Write</td>
<td>client</td>
<td>Write data to a bitfile</td>
</tr>
<tr>
<td>bfs_BitfileGetAttrs</td>
<td>SSM, client</td>
<td>Get bitfile managed object attributes</td>
</tr>
<tr>
<td>bfs_BitfileSetAttrs</td>
<td>SSM, client</td>
<td>Set bitfile managed object attributes</td>
</tr>
<tr>
<td>bfs_BitfileOpenGetAttrs</td>
<td>SSM, client</td>
<td>Get bitfile managed object attributes (for an open bitfile)</td>
</tr>
<tr>
<td>bfs_BitfileOpenSetattrs</td>
<td>SSM, client</td>
<td>Set bitfile managed object attributes (for an open bitfile)</td>
</tr>
<tr>
<td>bfs_ServerGetAttrs</td>
<td>SSM, client</td>
<td>Get (common) server managed object attributes</td>
</tr>
<tr>
<td>bfs_ServerSetAttrs</td>
<td>SSM, client</td>
<td>Set (common) server managed object attributes</td>
</tr>
<tr>
<td>bfs_Copy</td>
<td>Migration, client</td>
<td>Copy storage segments for a bitfile to the next storage hierarchy level</td>
</tr>
<tr>
<td>bfs_Copy</td>
<td>Migration, client</td>
<td>Move storage segments for a bitfile to the next storage hierarchy level</td>
</tr>
<tr>
<td>bfs_Purge</td>
<td>Purge, client</td>
<td>Reclaim space (i.e. purge segments) occupied by a bitfile</td>
</tr>
</tbody>
</table>
The Storage Server provides *APIs* for the following operations:

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ss_BeginSession</td>
<td>BFS, SSM, client</td>
<td>Start a storage server session</td>
</tr>
<tr>
<td>ss_EndSession</td>
<td>BFS, SSM, client</td>
<td>End a storage server session</td>
</tr>
<tr>
<td>ss_SSCreate</td>
<td>BFS, client</td>
<td>Create a storage segment</td>
</tr>
<tr>
<td>ss_SSUnlink</td>
<td>BFS, client</td>
<td>Delete a storage segment</td>
</tr>
<tr>
<td>ss_SSRead</td>
<td>BFS, client</td>
<td>Read data from a storage segment</td>
</tr>
<tr>
<td>ss_SSWrite</td>
<td>BFS, client</td>
<td>Write data to a storage segment</td>
</tr>
<tr>
<td>ss_SSGetAttrs</td>
<td>BFS, client</td>
<td>Get storage segment managed object attributes</td>
</tr>
<tr>
<td>ss_SSSetAttrss</td>
<td>BFS, SSM, client</td>
<td>Set storage segment managed object attributes</td>
</tr>
<tr>
<td>ss_SSMount</td>
<td>Migrate, Repack, SSM, client</td>
<td>Mount a storage segment and assign it to a session</td>
</tr>
<tr>
<td>ss_SSUnmount</td>
<td>Migrate, Repack, SSM, client</td>
<td>Unmount a storage segment</td>
</tr>
<tr>
<td>ss_SSCopySegment</td>
<td>Migrate, SSM, client</td>
<td>Copy storage segment to new segment on different virtual volume</td>
</tr>
<tr>
<td>ss_SSMoveSegment</td>
<td>Migrate, Repack, SSM, client</td>
<td>Move storage segment to new virtual volume</td>
</tr>
<tr>
<td>ss_MapCreate</td>
<td>SSM, client</td>
<td>Create storage map for a virtual volume</td>
</tr>
<tr>
<td>ss_MapDelete</td>
<td>SSM, client</td>
<td>Delete storage map for a virtual volume</td>
</tr>
<tr>
<td>ss_MapGetAttrss</td>
<td>SSM, client</td>
<td>Get storage map managed object attributes</td>
</tr>
<tr>
<td>ss_MapSetAttrss</td>
<td>SSM, client</td>
<td>Set storage map managed object attributes</td>
</tr>
<tr>
<td>ss_VVCreate</td>
<td>SSM, client</td>
<td>Create a virtual volume</td>
</tr>
<tr>
<td>ss_VVDelete</td>
<td>SSM, client</td>
<td>Delete a virtual volume</td>
</tr>
<tr>
<td>ss_VVMount</td>
<td>SSM, client</td>
<td>Mount a virtual volume</td>
</tr>
<tr>
<td>ss_VVUnmount</td>
<td>SSM, client</td>
<td>Unmount a virtual volume</td>
</tr>
<tr>
<td>ss_VVRead</td>
<td>SSM, client</td>
<td>Read a virtual volume</td>
</tr>
<tr>
<td>ss_VVWrite</td>
<td>SSM, client</td>
<td>Write a virtual volume</td>
</tr>
<tr>
<td>ss_VVGetAttrss</td>
<td>SSM, client</td>
<td>Get virtual volume managed object attributes</td>
</tr>
<tr>
<td>ss_NVSetAttrss</td>
<td>SSM, client</td>
<td>Set virtual volume managed object attributes</td>
</tr>
<tr>
<td>ss_PVCreate</td>
<td>SSM, client</td>
<td>Create a physical volume</td>
</tr>
<tr>
<td>ss_PVDelete</td>
<td>SSM, client</td>
<td>Delete a physical volume</td>
</tr>
<tr>
<td>ss_PVMount</td>
<td>SSM, client</td>
<td>Mount a physical volume</td>
</tr>
<tr>
<td>ss_PVUnmount</td>
<td>SSM, client</td>
<td>Unmount a physical volume</td>
</tr>
<tr>
<td>ss_PVRead</td>
<td>SSM, client</td>
<td>Read a physical volume</td>
</tr>
<tr>
<td>ss_PVWrite</td>
<td>SSM, client</td>
<td>Write a physical volume</td>
</tr>
<tr>
<td>ss_PVGetAttrss</td>
<td>SSM, client</td>
<td>Get physical volume managed object attributes</td>
</tr>
<tr>
<td>ss_PVsetAttrss</td>
<td>SSM, client</td>
<td>Set physical volume managed object attributes</td>
</tr>
<tr>
<td>ss_SSrvGetAttrss</td>
<td>SSM, client</td>
<td>Get Storage Server specific managed object attributes</td>
</tr>
<tr>
<td>ss_SSrvSetAttrss</td>
<td>SSM, client</td>
<td>Set Storage Server specific managed object attributes</td>
</tr>
<tr>
<td>ss_ServerGetAttrss</td>
<td>SSM, client</td>
<td>Get (common) server managed object attributes</td>
</tr>
<tr>
<td>ss_ServerSetAttrss</td>
<td>SSM, client</td>
<td>Set (common) server managed object attributes</td>
</tr>
</tbody>
</table>
The Mover provides APIs for the following operations:

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mvr_Read</td>
<td>SS, PVL, client</td>
<td>Read data from a device or devices</td>
</tr>
<tr>
<td>mvr_Write</td>
<td>SS, PVL, client</td>
<td>Write data to a device or devices</td>
</tr>
<tr>
<td>mvr_DeviceSpec</td>
<td>SS, client</td>
<td>Load a physical volume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Unload a physical volume</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Load message to device's display area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flush data to media</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write tape mark</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Read media label</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Write media label</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clear portion of disk</td>
</tr>
<tr>
<td>mvr_DeviceGetAttrs</td>
<td>SS, SSM, client</td>
<td>Get Mover device managed object attributes</td>
</tr>
<tr>
<td>mvr_DeviceSetAttrs</td>
<td>SS, SSM, client</td>
<td>Set Mover device managed object attributes</td>
</tr>
<tr>
<td>mvr_MvrGetAttrs</td>
<td>SSM, client</td>
<td>Get Mover specific managed object attributes</td>
</tr>
<tr>
<td>mvr_MvrSetAttrs</td>
<td>SSM, client</td>
<td>Set Mover specific managed object attributes</td>
</tr>
<tr>
<td>mvr_ServerGetAttrs</td>
<td>SSM, client</td>
<td>Get (common) server managed object attributes</td>
</tr>
<tr>
<td>mvr_ServerSetAttrs</td>
<td>SSM, client</td>
<td>Set (common) server managed object attributes</td>
</tr>
</tbody>
</table>

The Physical Volume Library provides APIs for the following operations:

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvl_Mount</td>
<td>client</td>
<td>Synchronously mount a single volume</td>
</tr>
<tr>
<td>pvl_MountNew</td>
<td>SS, client</td>
<td>Begin creating a set of volumes to atomically mount</td>
</tr>
<tr>
<td>pvl_MountAdd</td>
<td>SS, client</td>
<td>Add a volume to the set of volumes to be mounted</td>
</tr>
<tr>
<td>pvl_MountCommit</td>
<td>SS, client</td>
<td>Mount a set of volumes</td>
</tr>
<tr>
<td>pvl_MountCompleted</td>
<td>PVR</td>
<td>Notify the PVL a pending mount has completed</td>
</tr>
<tr>
<td>pvl_CancelAllJobs</td>
<td>SS, SSM, client</td>
<td>Cancel all jobs associated with a connection handle</td>
</tr>
<tr>
<td>pvl_DismountJobId</td>
<td>SS, SSM, client</td>
<td>Dismount all volumes associated with a specific job</td>
</tr>
<tr>
<td>pvl_DismountVolume</td>
<td>SS, SSM, client</td>
<td>Dismounts a single volume</td>
</tr>
<tr>
<td>pvl_DismountDrive</td>
<td>SSM, client</td>
<td>Forces the dismount of a specified drive</td>
</tr>
<tr>
<td>pvl_Import</td>
<td>SSM, client</td>
<td>Imports a new cartridge into HPSS</td>
</tr>
<tr>
<td>pvl_Export</td>
<td>SSM, client</td>
<td>Exports a cartridge from HPSS</td>
</tr>
<tr>
<td>pvl_Move</td>
<td>SSM, client</td>
<td>Move a cartridge from one PVR to another</td>
</tr>
<tr>
<td>pvl_NotifyCartridge</td>
<td>PVR</td>
<td>Notify the PVL that a cartridge has been check in or out of a PVR</td>
</tr>
<tr>
<td>pvl_WriteVolumeLabel</td>
<td>SS, SSM, client</td>
<td>Rewrite the internal label of a specified volume</td>
</tr>
<tr>
<td>pvl_AllocateVol</td>
<td>SS, SSM, client</td>
<td>Allocate a volume to a particular client</td>
</tr>
<tr>
<td>pvl_ScratchVol</td>
<td>SS, SSM, client</td>
<td>Return a volume to the scratch pool</td>
</tr>
<tr>
<td>pvl_DriveGetAttrs</td>
<td>SSM, client</td>
<td>Get drive managed object attributes</td>
</tr>
<tr>
<td>pvl_DriveSetAttrs</td>
<td>SSM, client</td>
<td>Set drive managed object attributes</td>
</tr>
<tr>
<td>pvl_VolumeGetAttrs</td>
<td>SSM, client</td>
<td>Get volume managed object attributes</td>
</tr>
<tr>
<td>pvl_VolumeSetAttrs</td>
<td>SSM, client</td>
<td>Set volume managed object attributes</td>
</tr>
<tr>
<td>pvl_QueueGetAttrs</td>
<td>SSM, client</td>
<td>Get PVL request queue managed object attributes</td>
</tr>
<tr>
<td>pvl_QueueSetAttrs</td>
<td>SSM, client</td>
<td>Set PVL request queue managed object attributes</td>
</tr>
<tr>
<td>pvl_RequestGetAttrs</td>
<td>SSM, client</td>
<td>Get PVL request queue entry managed object attributes</td>
</tr>
<tr>
<td>pvl_RequestSetAttrs</td>
<td>SSM, client</td>
<td>Set PVL request queue entry managed object attributes</td>
</tr>
<tr>
<td>pvl_PVLGetAttrs</td>
<td>SSM, client</td>
<td>Get PVL specific managed object attributes</td>
</tr>
<tr>
<td>pvl_PVLSetAttrs</td>
<td>SSM, client</td>
<td>Set PVL specific managed object attributes</td>
</tr>
<tr>
<td>pvl_ServerGetAttrs</td>
<td>SSM, client</td>
<td>Get (common) server managed object attributes</td>
</tr>
<tr>
<td>pvl_ServerSetAttrs</td>
<td>SSM, client</td>
<td>Set (common) server managed object attributes</td>
</tr>
</tbody>
</table>
The Physical Volume Repository provides APIs for the following:

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pvr_Mount</td>
<td>PVL, client</td>
<td>Asynchronously mount a single volume</td>
</tr>
<tr>
<td>pvr_MountComplete</td>
<td>PVL, client</td>
<td>Notify PVL a requested mount has completed</td>
</tr>
<tr>
<td>pvr_DismountCart</td>
<td>PVL, client</td>
<td>Dismount a single cartridge</td>
</tr>
<tr>
<td>pvr_DismountDrive</td>
<td>PVL, client</td>
<td>Dismount the cartridge in a given drive</td>
</tr>
<tr>
<td>pvr_Inject</td>
<td>PVL, SSM, client</td>
<td>Accept a new cartridge into the PVR</td>
</tr>
<tr>
<td>pvr_Eject</td>
<td>PVL, SSM, client</td>
<td>Eject a cartridge from the PVR</td>
</tr>
<tr>
<td>pvr_Audit</td>
<td>SSM, client</td>
<td>Audit all or part of a repository checking external cartridge labels when possible</td>
</tr>
<tr>
<td>pvr_LocateCartridge</td>
<td>PVL, client</td>
<td>Verify whether or not a PVR manages a cartridge</td>
</tr>
<tr>
<td>pvr_SetDrive</td>
<td>PVL, client</td>
<td>Takes drives in the PVR on-line or off-line</td>
</tr>
<tr>
<td>pvr_CartridgeGetAttrs</td>
<td>SSM, client</td>
<td>Get a cartridge managed object attributes</td>
</tr>
<tr>
<td>pvr_CartridgeSetAttr</td>
<td>SSM, client</td>
<td>Set a cartridge managed object attributes</td>
</tr>
<tr>
<td>pvr_PVGetAttrs</td>
<td>SSM, client</td>
<td>Get PVR specific managed object attributes</td>
</tr>
<tr>
<td>pvr_PVRSetAttr</td>
<td>SSM, client</td>
<td>Set PVR specific managed object attributes</td>
</tr>
<tr>
<td>pvr_ServerGetAttr</td>
<td>SSM, client</td>
<td>Get (common) server managed object attributes</td>
</tr>
<tr>
<td>pvr_ServerSetAttr</td>
<td>SSM, client</td>
<td>Set (common) server managed object attributes</td>
</tr>
<tr>
<td>pvr_ListPendingMounts</td>
<td>SSM, client</td>
<td>List all currently pending mounts for the PVR</td>
</tr>
</tbody>
</table>

The Storage System Manager provides APIs for the following operations:

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ssm_Adm</td>
<td>client</td>
<td>Perform administrative request on one or more servers (shutdown, halt, mark down, reinitialize, start)</td>
</tr>
<tr>
<td>ssm_AttrGet</td>
<td>client</td>
<td>Get managed object attributes</td>
</tr>
<tr>
<td>ssm_AttrReg</td>
<td>client</td>
<td>Register an SSM client to receive notifications of data change in managed objects</td>
</tr>
<tr>
<td>ssm_AttrSet</td>
<td>client</td>
<td>Set managed object attributes</td>
</tr>
<tr>
<td>ssm_Checkin</td>
<td>client</td>
<td>Accept checkins from data server clients</td>
</tr>
<tr>
<td>ssm_Checkout</td>
<td>client</td>
<td>Accept checkouts from data server clients</td>
</tr>
<tr>
<td>ssm_ConfigAdd</td>
<td>client</td>
<td>Add a new entry to a configuration file</td>
</tr>
<tr>
<td>ssm_ConfigDelete</td>
<td>client</td>
<td>Delete an entry from a configuration file</td>
</tr>
<tr>
<td>ssm_ConfigUpdate</td>
<td>client</td>
<td>Update a configuration file entry</td>
</tr>
<tr>
<td>ssm_Delog</td>
<td>client</td>
<td>Allow accept to the delog command</td>
</tr>
<tr>
<td>ssm_DriveDismount</td>
<td>client</td>
<td>Dismount a drive</td>
</tr>
<tr>
<td>ssm_JobCancel</td>
<td>client</td>
<td>Cancel a Physical Volume Library job</td>
</tr>
<tr>
<td>ssm_CartImport</td>
<td>client</td>
<td>Import cartridges into the Physical Volume Library</td>
</tr>
<tr>
<td>ssm_CartExport</td>
<td>client</td>
<td>Export cartridges from the Physical Volume Library</td>
</tr>
<tr>
<td>ssm_ResourceCreate</td>
<td>client</td>
<td>Create resources (physical volume, virtual volume, and storage map) in the Storage Server</td>
</tr>
<tr>
<td>ssm_ResourceDelete</td>
<td>client</td>
<td>Delete resources (physical volume, virtual volume, and storage map) from the Storage Server</td>
</tr>
<tr>
<td>ssm_AlarmNotify</td>
<td>Logging</td>
<td>Receive notifications of alarms</td>
</tr>
<tr>
<td>ssm_EventNotify</td>
<td>Logging</td>
<td>Receive notifications of events</td>
</tr>
<tr>
<td>ssm_MountNotify</td>
<td>PVL</td>
<td>Receive notifications of tape mounts and dismounts</td>
</tr>
<tr>
<td>ssm_BitfileNotify</td>
<td>BFS</td>
<td>Receive bitfile data change notifications</td>
</tr>
<tr>
<td>ssm_CartNotify</td>
<td>PVR</td>
<td>Receive cartridge data change notifications</td>
</tr>
<tr>
<td>ssm_DeviceNotify</td>
<td>PVL</td>
<td>Receive device data change notifications</td>
</tr>
<tr>
<td>ssm_DriveNotify</td>
<td>PVL</td>
<td>Receive drive data change notifications</td>
</tr>
<tr>
<td>ssm_LogfileNotify</td>
<td>Logging</td>
<td>Receive log file data change notifications</td>
</tr>
<tr>
<td>ssm_MVRNotify</td>
<td>Mvr</td>
<td>Receive Mover specific data change notifications</td>
</tr>
<tr>
<td>ssm_MapNotify</td>
<td>SS</td>
<td>Receive storage map data change notifications</td>
</tr>
</tbody>
</table>
The following managed objects have attributes which may be queried (and set) by SSM:

<table>
<thead>
<tr>
<th>Managed Object</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name Server</td>
<td>Volume</td>
</tr>
<tr>
<td>Bitfiles</td>
<td>Physical Volume Library queue</td>
</tr>
<tr>
<td>Bitfile Server (common)</td>
<td>Physical Volume Library request entry</td>
</tr>
<tr>
<td>Storage segments</td>
<td>Physical Volume Library Server (common)</td>
</tr>
<tr>
<td>Storage maps</td>
<td>Cartridge</td>
</tr>
<tr>
<td>Virtual volumes</td>
<td>Physical Volume Repository server specific</td>
</tr>
<tr>
<td>Physical volumes</td>
<td>Physical Volume Repository Server (common)</td>
</tr>
<tr>
<td>Storage Server specific</td>
<td>Security server</td>
</tr>
<tr>
<td>Storage Server (common)</td>
<td>Log Daemon server (common)</td>
</tr>
<tr>
<td>Mover device</td>
<td>Log Client server (common)</td>
</tr>
<tr>
<td>Mover server specific</td>
<td>Structured File Server</td>
</tr>
<tr>
<td>Mover server (common)</td>
<td></td>
</tr>
<tr>
<td>Drive</td>
<td></td>
</tr>
</tbody>
</table>

The Storage System Manager also receives the following type of notifications from the HPSS server components:

<table>
<thead>
<tr>
<th>Notifications</th>
<th>Tape mounts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alarms</td>
<td>Data changes for registered object attributes</td>
</tr>
<tr>
<td>Events</td>
<td></td>
</tr>
</tbody>
</table>

Some of the more important management operations which may be performed by the Storage System Manager include:

<table>
<thead>
<tr>
<th>Operations</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Import / create resources</td>
<td>Repack</td>
</tr>
<tr>
<td>Import cartridges</td>
<td>Delog</td>
</tr>
<tr>
<td>Export cartridges</td>
<td>Set devices online / offline</td>
</tr>
<tr>
<td>Move cartridges (from one PVR to another)</td>
<td>Dismount drive</td>
</tr>
<tr>
<td>Audit PVR</td>
<td>Start / stop / reinitialize / halt servers</td>
</tr>
</tbody>
</table>
Migration / Purge provides APIs for the following operations:

<table>
<thead>
<tr>
<th>API</th>
<th>Clients</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>migr_StartMigration</td>
<td>SSM, client</td>
<td>Start migration for a particular storage class</td>
</tr>
<tr>
<td>migr_StartPurge</td>
<td>SSM, client</td>
<td>Start purge for a particular storage class</td>
</tr>
<tr>
<td>migr_MPSSetAttrs</td>
<td>SSM, client</td>
<td>Get the migration-purge server attributes</td>
</tr>
<tr>
<td>migr_MPSSetAttrs</td>
<td>SSM, client</td>
<td>Set the migration-purge server attributes</td>
</tr>
<tr>
<td>migr_ServerGetAttrs</td>
<td>SSM, client</td>
<td>Get (common) server managed object attributes</td>
</tr>
<tr>
<td>migr_ServerSetAttrs</td>
<td>SSM, client</td>
<td>Set (common) server managed object attributes</td>
</tr>
</tbody>
</table>
BUILDING PETABYTE DATABASES WITH OBJECTIVITY/DB

Leon Guzenda
Objectivity, Inc., Mountain View, California, USA.

Abstract
Objectivity, Inc. has been working with the CERN RD45 Project for several years to investigate the possibility of using a commercial Object Database Management System (ODBMS) as the basis for storing data from the Large Hadron Collider and other projects. This paper explains the fundamental concepts behind Objectivity/DB in the context of the problems inherent in building, using and maintaining multi-petabyte databases.

1. BACKGROUND
Since its inception in 1988, Objectivity, Inc. has concentrated its efforts in two main markets: becoming embedded in a wide range of specialized software and hardware products; and large projects, primarily in the telecom and Very Large Database (VLDB) arenas.

In 1997 the CERN RD45 Project selected Objectivity/DB for further study as a candidate for storing data from the LHC and other HEP projects. The desirability of moving to an object-oriented programming environment had quickly shown the need for a mechanism other than structured files or RDBMSs for storing and manipulating the data. Several projects are now moving into production with Objectivity/DB. This paper looks at the main challenges inherent in building multi-petabyte databases, some current solutions and some future work.

2. THE CHALLENGES
The most obvious feature of the data from large HEP projects is the sheer volume of data produced. Much of the raw data is essentially simple, or is distilled into commonly understood representations. It is not currently economically feasible to keep all of this data, or even all of the distilled data, online at all times. This presents a problem of tape management that is rarely seen outside of a few commercial data warehouses.

Storing the data is one problem. Manipulating it efficiently in compute intensive algorithms puts an extra stress on the DBMS that is similar to the requirements of advanced CAD applications. The DBMS has to attempt to access data at speeds comparable to those provided by RAM, yet it is confronted with disks that typically have 5 to 15 mSec response times. There is a 10 ** 6 disparity between RAM and disk access and this gap may widen as the years progress.

Once large bodies of data have been collected from repetitive or similar experiments, there remains the problem of being able to make ad hoc searches across the data. This problem is not unique to the HEP community. Multidimensional searches are equally important to the data warehousing community and to other physical and biological experimenters.

There is both a need to gather and store data over very long periods (25 years, or more) at virtual central locations and a need to keep isolated data stores that may be added to the “central” store at an appropriate time. Similarly, it is convenient to be able to extract datasets for local (private) processing. These local datasets may or may not be returned to the “central” database.
3. CURRENT SOLUTIONS

3.1 Scalability

Objectivity/DB is a scalable, distributed Object DBMS. It is scalable both in the number of users that may be storing or manipulating data at a given time and in the total amount of data that may be addressed in a single transaction.

Objectivity/DB holds objects (C++, Java or Smalltalk) in standard files, which are internally and individually structured, into a hierarchical storage structure. The highest level of this structure is the Federated Database (often simply known as the Federation). The Federated Database holds schemas, catalogs, security information and housekeeping for backups. This information may be replicated into Autonomous Partitions. A Partition is a logical subset of the Federation. Each Partition controls one or more Databases. These Databases may be on different machines and may hold objects created by heterogeneous operating system, platform or compiler/language environments. Each Database may itself be replicated across many Partitions (no more than one image per Partition). The combination of Partitions, with their own system resources, and Database Replicas provides a very fault tolerant environment.

Each Database is further sub-divided into one or more Containers. A Container may hold one or more Objects, which need not be of the same class. Each Object looks like a standard C++, Java or Smalltalk object with a few extensions to cover varying length arrays (VArrays) and named relationships between object and/or containers (Associations).

Each object is given a 64-bit Object Identifier (OID) which is retained for its lifetime. On current filesystems, the OID can locate an object anywhere within a 64 Exabyte ($10^{18}$) byte address space. Users never manipulate OIDs directly. They use REFS, which are smart pointers, to refer to an existing object.

The data is transferred to and from disk by local or remote Page Servers. These may be industry standard nfsd daemons, making it possible to use a wide range of file servers, or Objectivity’s Advanced Multithreaded Server (AMS). A catalog of databases (kept in the Federated Database and synchronized across Partitions) makes it possible for the Storage Manager, which runs in the client process space, to locate any object. This distributed and parallel architecture means that unrelated processes or threads may use their own processing and I/O resources without interfering with one another. As an example, two users may be running applications on their own laptop PCs, totally disconnected from the rest of the federation. Or, two powerful workstations within the same LAN or WAN may be caching local copies of read-only data while creating other data, eventually to be shared, from that data. This is particularly important in the data acquisition phase of a typical project. Individual processors may write to dedicated disks and maintain a very high throughput. They need not be constrained by a central server.

The AMS may be interfaced with the High Performance Storage System (HPSS) via a layer called oofs. This layer can be re-implemented by users to allow it to access non-HPSS equipment. HPSS is an implementation of an IEEE standard Mass Storage Device controller. It holds a great deal of promise for managing the physical media involved in petabyte-plus systems.

Concurrency is handled with the traditional ACID transaction paradigm, implemented via simple Lock Server processes, which provide hierarchical Gray-code locking. Each Partition has its own Lock Server, to avoid having single points of failure in the Federation. The Lock Servers handle distributed deadlock detection.

3.2 Fast Access to the Data

Objectivity/DB provides several standard mechanisms for finding data rapidly. These include B-tree indices, hash tables, collection classes (C++ STL bags, sets, lists etc.) and navigational access via named associations. Objects may also be named and versioned. Objectivity/DB provides iterators for
finding groups of data. The iterators may be initialised with a simple or compound predicate (which may even be a valid SQL SELECT statement!). There are standard iterators for moving down the containment hierarchy, scanning a layer (or nested layers) of the hierarchy; and for traversing named associations. The main difference between an iterator and the SELECT plus move cursor arrangement provided by SQL is that an iterator starts returning qualified objects as soon as any are found. This is obviously useful if extremely large numbers of objects are to be examined. An RDBMS will generally assemble a complete view or table to satisfy the query before sending any response back to the client.

Objectivity/DB provides both uni-directional and bi-directional associations. Their cardinality may be 1:1, 1:N, N:1 or M:N. Associations may link objects in different containers or databases. This is transparent to the user, as Objectivity/DB provides a single address space within the federation.

It is often feasible to cluster data that will generally be used together into the same logical/physical pages on disk. This is done by providing a clustering directive at the time that an object is created. The clustering directive may be a REF of an object, container or database; or it may be a function that implements some policy. It may even be declustering rapidly arriving objects onto different disk drives to keep up with the incoming stream. It is possible to recluster objects at a later date, using the moveObject() method. This changes the OID, but all bi-directional associations, indices, hash-tables etc. are automatically adjusted. This mechanism was chosen in preference to a logically separate table of OIDs in order to reduce storage overheads and performance bottlenecks. It is interesting to note that immediately after committing a newly created object to disk there is no single place that holds the actual OID of the object. This means that a semi-smart process working through physical pages could theoretically reconstruct a badly corrupted database.

The physical page size is set by the database designer and is the basic unit of transfer between disk and RAM and across the network. Objectivity/DB differentiates between pages with many small objects in them and objects which span many physical pages. It only transfers the pages that are actually used into the client’s cache. As a process or thread acquires locks to access each container of objects it picks up information about the status of its previously cached pages for that container. In good cases, the previously cached pages have not been changed by any other thread/process, so they may be reused. This gives a considerable performance boost in a surprisingly wide range set of circumstances. In the worst case, out of date pages are refreshed automatically by the Storage Manager, so cache management is not a burden to the application programmer.

We decided early on in the development of Objectivity/DB that it would be too expensive to lock individual applications. Experiments confirmed our view, particularly with large sets of complex data and relationships. Queuing and deadlocks impacted performance and complicated the error recovery in individual methods and programs. So, we decided to lock at the container level. After all, the container stores objects that are frequently accessed together, so maybe we can exploit this behaviour. We implemented an extra transaction mode, called MROW, for Multi-Reader, One Writer to overcome the objection that too many objects would be made inaccessible if an updater locks a container for an extended period. MROW allows a writer to create a new version of a container while readers view the currently committed version. This mechanism reduces lock requests (which typically run at around 400 RPCs per second across an Ethernet), reduces queuing and deadlocks; and increases system throughput. The outcome is that object level locking is still low on the list of user group requirements after almost ten years!

3.3 Ad Hoc Queries

As mentioned above, Objectivity/DB provides iterators, which may be initialised with simple or compound predicate conditions. The iterators will take advantage of any relevant indices. The Object Database Management Group included an Object Query Language as a part of the ODMG’93 standard for Object Databases. After careful consideration, Objectivity decided to stick with our commitment to provide both an ANSI standard SQL 2 interface to Objectivity, called Objectivity/SQL++ rather than try to implement the parsers and APIs of OQL. We also added ODBC
interfaces. Objectivity/SQL++, released with Objectivity Version 2.0, became the first implementation of standard SQL based on a pure ODBMS. The strategy has paid off. There are over 600 ODBC-compliant tools that will work with RDBMSs and Objectivity/SQL++. There are no independent (i.e. provided by a vendor other than an ODBMS vendor) OQL tools on the market today, after five years of potential market development. So, it now looks likely that OQL will converge with SQL3 and 4, ratifying our original strategy.

The current release of Objectivity/SQL++ can execute simple methods within the SQL++ server as a part of a query. For instance, it could execute a getAge() method to compute a value from a Date_of_Birth field in a Person object. Later, as the syntax for doing this within SQL4 becomes stable, we hope to extend SQL++ to handle any existing C++, Java or Smalltalk method/message.

3.4 Distributed and Private Databases

The Federation plus Partition and replication mechanism is a very powerful paradigm for providing a distributed data store within a collaboration. Everything within a federation is potentially visible from anywhere on the network. Individuals may create their own Partitions and replicate any databases they wish. They may also create new databases and then either delete them or make the available to the community. It is also possible to copy a database and attach it to a compatible Federation. It must retain the same database name and identifier and the schemas must be identical for this to be successful.

C++ environments are currently compiled and programs consist of binary images. Objectivity/DB must present the C++ objects in the format expected by the local compiler. So, if a class definition changes there are immediate ramifications across the system. Programs must be recompiled and object instances must be treated appropriately. Objectivity/DB provides a variety of ways of dealing with the object instances. For any class that is changed, the application designer may opt to have all instances atomically migrated to the new format; or have instances migrated as they are located; or have instances converted every time they are encountered. This still leaves the problem of migrating the C++ binary forward, but this is a code control problem with or without Objectivity/DB being present.

Given the pace of technological developments and the life span of some of the projects, it is completely impracticable to assume that the hardware, operating systems and languages will be constant throughout a project. Fortunately, Objectivity/DB transparently converts objects on demand to help cope with these issues. It does this by building a state table at the time that the object classes are first defined to Objectivity/DB (through C++ DDL or Java/Smalltalk environments). Only the objects or VArrays that are actually used are converted. The state machine only touches the fields that need conversion or movement. This is more efficient than adopting a single neutral format, which necessitates double conversion of almost all objects whenever they are actually used. This means that a few CPU cycles may be used, but the time taken is insignificant compared with the I/O which has immediately preceded the first conversion.

4. FUTURE WORK

We have provided named associations, hash tables and multi-key B-tree indices. We are currently experimenting with a multi-dimensional R-tree geospatial index. If this proves successful then it may be generalised for other markets. We are also tracking work by researchers at CERN and various Universities around the world to provide a wider variety of indexing techniques as a standard part of the product.

The initial experiments with data replication have been quite encouraging. A database was replicated between CERN (Geneva) and CalTech (Pasadena, California) and subsequent updates were successfully applied either immediately or by automatic resynchronization. However, we will
continue to provide improvements in this area, for example, by multicasting pages into the network whenever possible.

The current release of Objectivity (V5) only supports one file per database. This will be changed at the next release to meet the architectural requirement that a file may hold one or more containers, each container coming from the same database. We also need to remove the restriction that a copied database may only use its existing name and OID if it is attached to an existing Federation.

Initial work with HPSS has been encouraging. However, there is further work to do in order to provide a random access disk cache and to implement Kerberos authentication. HPSS has a powerful concept of policies, or “Class of Service” for managing the staging of files. This looks very promising. We are seeking ways to exploit this mechanism, possibly by sending hints between the client and the AMS, or between the AMS and the HPSS.

4.1 References


Abstract
Starting from 2005, the LHC experiments will generate an unprecedented amount of data. Some 100 Peta-Bytes of event, calibration and analysis data will be stored and have to be analysed in a world-wide distributed environment. At CERN the RD45 project has been set-up in 1995 to investigate different approaches to solve the data storage problems at LHC. The focus of RD45 soon moved to the use of Object Database Management Systems (ODBMS) as a central component. This paper gives an overview of the main advantages of ODBMS systems for HEP data stores. Several prototype and production applications will be discussed and a summary of the current use of ODBMS based systems in HEP will be presented. The second part will concentrate on physics data analysis based on an ODBMS.

1 INTRODUCTION
1.1 Data Management at LHC
The new experiments at the Large Hadron Collider (LHC) at CERN will gather an unprecedented amount of data. Starting from 2005 each of the four LHC experiments ALICE, ATLAS, CMS and LHCb will measure of the order of 1 Peta Byte ($10^{15}$ Bytes) per year. All together the experiments will store and repeatedly analyse some 100 PB of data during their lifetimes. Such an enormous task can only be accomplished by large international collaborations. Thousands of physicists from hundreds of institutes world-wide will participate. This also implies that nearly any available hardware platform will be used resulting in a truly heterogeneous and distributed system.

The computing technical proposals of LHC experiments do not only require access to the data from remote sites but in addition ask for distribution of the data store itself to several regional centers.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Data Rate</th>
<th>Data Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALICE</td>
<td>1.5 GB/sec</td>
<td>1 PB/month (1 month per year)</td>
</tr>
<tr>
<td>ATLAS</td>
<td>100 MB/sec</td>
<td>1 PB/year</td>
</tr>
<tr>
<td>CMS</td>
<td>100 MB/sec</td>
<td>1 PB/year</td>
</tr>
<tr>
<td>LHCb</td>
<td>400 TB/year</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Expected Data Rates and Volumes at LHC

1.1.1 HEP Data Models
HEP data models are typically very complex. The number of different data types (e.g. bank types or classes) needed to describe the event data of a large HEP experiment reaches easily several hundreds.
A single event often contains thousands of individual data structures (e.g. banks or objects) and a large number of relations between those data items (e.g. links or pointers).

The design and documentation of HEP data models is an essential and non-trivial task during the experiment software development process. Especially the data model of stored data structures is of significant influence on many other software developments. Since the definition of this data is shared between multiple subsystems (e.g. data acquisition, event reconstruction and physics analysis) with very different access patterns, it is often difficult to fulfil all flexibility and performance requirements in a single design.

All LHC experiments exploit Object Oriented (OO) technology to implement and maintain their very large software systems. Today most software development is done in C++ with a growing interest in Java. The data store therefore has to support the main concepts of these OO languages such as abstraction, inheritance, polymorphism and parameterised types.

1.1.2 The RD45 project

From the analysis of the LHC data management requirements, it seemed clear that existing solutions based on FORTRAN and sequential files such as ZEBRA would be inadequate for the LHC era. At CERN, the RD45 project was started in 1995 to investigate new solutions to the LHC data management problems. After an evaluation of different technology choices such as language extensions for persistency, light-weight object managers, object request brokers, RD45 focused rapidly on a solution consisting of a commercial Object Database Management System (ODBMS) coupled to a Mass Storage System (MSS).

2 OBJECT DATABASE SYSTEMS

2.1 ODBMS and Programming Languages

The natural view of data for application programmers is that of a network of objects in the application memory space. Figure 1 shows a simple example of such a configuration describing part of the event data of some HEP experiment.

![Diagram](image)

**Figure 1: Simple Logical Model of Event Data Objects**

A large fraction of any programs operation will consist of navigation within this net to find needed objects, change their state, create new objects or delete existing ones, e.g. navigation from the event object to its tracking detector, retrieving a particular track from its track list, and navigation to all associated hit objects to perform a track refit.
For transient data - objects that only exist within the context of a single program - this navigation is well supported by OO languages. The pointer and reference types provided by C++ and Java allow efficiently creating and maintaining complex in-memory object networks. When I/O operations have to be performed on such a network, e.g. because object data need to be stored or to be exchanged between two programs, the support from today’s OO languages is rather limited.

2.1.1 Object Input / Output

Neither C++ nor Java provides an I/O sub-system capable of dealing directly with object trees or nets. The significant burden of providing this capability within a particular application is therefore left to the application programmer. In particular the programmer is left with the task to maintain two distinct copies of each data item that has to be stored:

- “in-memory” data – an object net with pointers or references describing relations between objects
- “on-disk” data – a sequence of bytes in one or more disk files

Since the representations of both copies are necessarily different (e.g. for heterogeneity reasons) the application code has to perform a quite complex transformation between these copies. In particular the application programmer has to explicitly code:

- When (and how often) to perform the transfer?
  The user has to explicitly trigger any data transfer (e.g. read data from disk before first use of some program variable, write to disk after its last update) and any re-transfer in case that the on disk data might have been changed by another program. Since in a complex HEP application it is difficult to predict which data items will be used, often simply all event data is transferred. This approach may result in degraded performance.

- How to perform the transformation?
  The network of objects has to be decomposed into single objects (or even single attributes) which may be stored directly. Any pointers or references have to be handled by special code that translates them into some storable format.

Often more than one third of the application code is necessary to perform this in principle well-defined mapping between disk and memory. In practice it turns out that maintaining this I/O related code is not only tedious but also relatively error prone. Many software problems in fact result from the lack of consistency between “in-memory” and “on-disk” data copies. Since the I/O subsystem is only loosely bound to the programming language, typically as a set of library routines, consistency cannot be maintained automatically. In addition the large fraction of I/O related code obscures the real services provided by a given class and makes it sometimes impossible to understand the objective of a code fragment without knowing about the details of the I/O related code.

2.1.2 Object Persistency

Object databases feature a very different I/O model. The starting point is a so-called “tight language binding” in which the consistency between data on disk and in memory is maintained by the database system. The programmer of an ODBMS application in fact only deals with a single copy of the data, the object in the programming language itself. The database system is responsible for maintaining the semantics of persistent objects: objects that retain all their state information between two program contexts.

Any data transfers needed to synchronise program objects with the data on disk are performed automatically by the ODBMS on a per object basis. Only data of those objects that are actually used
by a particular program are transferred which might result in a drastically increased application performance compared to reading an entire event.

ODBMSs transfer complete objects from disk into application memory. The state of all data members and the dynamic type of a persistent object are retained. Virtual function calls through a base class pointer behave as expected (support for polymorphism). ODBMSs fully support abstract data types and allow creating persistent types using templates.

### 2.2 Navigational Access

During the creation of a new persistent object the database assigns a unique Object Identifier (OID) to each object. When an object is accessed from an application program, its OID is used by the database to find the object data in the disk store. The different ODBMS products vary largely in their OID implementation ranging from direct extensions of virtual memory pointers (Objectstore) to structures that refer more directly to a physical location in the disk store (Objectivity/DB).

OID may themselves be embedded as data members in persistent objects, which allows implementing a relation between two persistent objects (association). Most ODBMS products allow creating in addition to uni-directional 1-to-1 relations 1-to-n associations (between one object and a varying number of other objects) and bi-directional associations.

#### 2.2.1 Smart Pointers

An application programmer typically does not use OID values directly but rather through so-called smart pointer types which allow to implement “on-demand“ I/O. Smart pointers are small objects which are provided by the database implementation that behave semantically as a normal object pointer. E.g. in C++ they allow to use the “->” operator to access an object attribute or to call a method. During this access the smart pointer will call back the database system to retrieve object data from disk if necessary. ODBMSs maintain an object cache in the application program (client side cache) to increase the performance of repeated accesses to the same objects.

```cpp
Collection<Event> events;       // an event collection
Collection<Event>::iterator evt;  // a collection iterator

// loop over all events in the input collection
for (evt = events.begin(); evt != events.end(); evt++)
{
    // access the first track in the tracklist
    d_Ref<Track> aTrack;
    aTrack = evt->tracker->tracks[0];

    // print the charge of all its hits
    for (int i = 0; i < aTrack->hits.size(); i++)
        cout << aTrack->hits[i]->charge << endl;
}
```

**Example 1: Navigation using a C++ program**

---

1 Some databases like Objectivity/DB treat large embedded attributes in an on-demand fashion. E.g. large embedded arrays will be read from disk only if the application accesses at least one array element.
As a consequence of the tight binding of ODBMS to the programming language the application programmer perceives the database as a natural extension of normal “in memory” objects. Using the database one can create networks of objects with indefinite lifetime and efficiently navigate among them.

2.3 Schema Handling

Before any instances of a persistent C++ class may be created, the class layout has to be registered with the database. The information about attribute position, name and type of attributes is used e.g., to provide the correct memory layout for an object on all different platforms.

For the C++ language this class registration is performed using a pre-processor program which scans class definitions of persistent classes in Objectivity’s Data Definition Language (DDL) and generates C++ header and implementation files for persistent classes. The generated header files define the class interface for clients of a persistent class. The generated implementation files contain C++ code which implements smart-pointer types and various collection iterators for each persistent class. All generated files are then compiled together with any other application code and linked against the Objectivity library to form a complete database application.

The set of all class definitions - also called database schema - is stored centrally in the federation file together with the catalogue of all database files.

![Diagram of Schema Capture and Build Process]

2.4 Consistent Access to Shared Data

ODBMS products provide support for multiple clients working on the same data store and concurrently updating it. Usually ODBMSs introduce a central “lockserver” that is responsible to coordinate the updates by keeping a lock table for the whole system. To guarantee data consistency in the
system, object databases use (as e.g. relational database) the notion of transactions. Any data change is part of a transaction with the ACID properties. Some vendors provide special transaction modes in which multiple reading processes can coexist with a single writer per locked entity which may enhance the concurrency behaviour. This feature is very useful for many HEP applications:

- **Simplified Support of Parallel Applications**
  Many sub-systems of LHC experiments like data acquisition, filter- and reconstruction farms and distributed simulation will only achieve their performance requirements by making use of parallel processing. These systems will profit from the build-in concurrency support of the data store.

- **Data Consistency and Reduced Storage Size**
  During HEP analysis the current practice involves many redundant copies of the original data. Starting from the fully reconstructed data one has to repeatedly create copies of the original data selecting a subset of the events, a subset of the data within an event or both. The reason for these redundant data copies is the inability of today’s I/O systems to effectively access sparse data, to re-cluster data or to add additional data to an existing store. In addition to the often significantly increased total storage size these data copies may lead to subtle consistency problems. Since any new reconstruction of the original events invalidates these copies, they have to be manually updated before results of different analysis groups can be compared.

2.5 **Physical Store Implementation**

All ODBMS products use a multilevel hierarchy to implement the possibly distributed physical store. Objectivity/DB for example uses a hierarchy of five different levels to implement the physical storage. The topmost level - the Federated Database - keeps system wide information about the shape of persistent classes within the store. In addition the catalogue of physical location of all data files is kept centrally in this file. Each federation consists of up to 64k databases - files that contain the actual data of all stored objects. Each database is structured internally into up to 32k “containers” - contiguous areas of objects within a database file. Containers consist themselves of up to 64k pages containing the actual object data. The starting position of object data on a page is called slot and uniquely defines a particular object on the page.

---

**Figure 3: Storage Hierarchy in Objectivity/DB**

The structure of the physical store hierarchy is directly reflected by the internal structure of the OID implementation. A 4-tuple of 16 bit numbers uniquely references any object within the store.
2.5.1 Separation of Logical and Physical Storage Model

The concept of OIDs allows to directly access any object in the potentially large distributed store without requiring the application programmer to maintain store implementation details like e.g. file and host names. Since this information about the physical layout of the store is kept only once, centrally by the database, it is much easier to change the storage layout without compromising existing applications. One may change the location of a particular file to a new host by moving the data and changing the catalogue entry. Since the catalogue is shared by all database applications, they will use the data from the new location without any change.

![Figure 4: Object Identifier Implementation used by Objectivity/DB](image)

2.5.2 Data Clustering and Re-Clustering

An important feature offered by several ODBMS products is the support of object clustering. When a persistent object is created, the user may supply information where the object should be placed within the physical storage hierarchy. In C++ a clustering hint may be passed as an argument to the new operator. The statement

```cpp
d_Ref<Track> aTrack = new(event) Track;
```

will for example instruct the database to create a new persistent track object physically close to the event object. This ability to cluster data on the physical storage medium is very important for optimising the performance of applications which access data selectively. The goal of this clustering optimisation is to transfer only useful data from disk to the application memory (or one storage level below: from tape storage to a disk pool). Grouping data close together

![Figure 5: Physical Storage Hierarchy and Logical User View](image)
that will later be read together can drastically reduce the number of I/O operations needed to acquire this data from disk or tape. It is important to note that this optimisation requires some knowledge about the relative contributions of different access patterns to the data.

An simple clustering strategy is the “type based clustering” where all objects of some particular class are placed together: e.g. Track and Hit objects within an event may be placed close to each other since both classes will often be used together during the event reconstruction.

For physics analysis this simple approach is probably not very efficient since the selection of data that will be read by a particular analysis application depends more on the physics process. In this case one may group the analysis data for a particular physics process together.

2.5.3 Architectural and Practical Storage Size Limitations:

The theoretical storage size constraints in the current Objectivity implementation result from the implementation object identifier in four 16-bit words. Each OID consists of four parts:

- **16 bit database number**
  - resulting in up to 64K databases per federated database
- **15 bit container number (one bit used internally)**
  - 32K containers per database
- **16 bit page number**
  - 64K logical pages per container
- **16 bit slot number**
  - 64K possible object locations per page

Assuming that the maximum number of database pages is allocated one obtains a maximum container size of 4GB (for 64kB page size) or 0.5GB (for 8kB page size). The theoretical limit for the total size of a federated database would amount in this case to some 10 000PB.

In the current implementation the maximum size of a federation is still significantly constrained by the maximum file size in the system. This calculation of a theoretical limit assumes database sizes of 128TB. Since in the current implementation each database is represented by a single file, such large databases are not practical. Assuming a maximum file size (and therefore maximum database size) of 100GB one obtains a maximum federation size of 6.5PB. Since this might be a limitation for LHC experiments, RD45 has requested to modify the mapping between database and physical files to allow multiple files per database.

2.6 Limits and Scalability Tests

Various tests have been performed to check the scalability of Objectivity/DB. Federated Databases of 0.5 TB have been demonstrated and multiple federations of 20-80GB are used today in production, some of them also exploiting the parallel I/O capabilities. The NA45 experiment for example performed their reconstruction and formatting on 32 filter nodes writing in parallel into a single federated database. Even more concurrent database clients have been simulated in a recent test performed at Caltech. On a HP Exemplar supercomputer up to 240 concurrent readers have been used successfully against a single federated database.

3 Objectivity Specific Features

In addition to the generic ODBMS functionality that is implemented by most vendors the products differ significantly in their data distribution and replication features. The following section describes these specific features and their potential use in HEP applications in more detail.
3.1 Federations of distributed databases

Applications are connected to one Objectivity federated database at a time. A federated database consists of many database files that may be located on different hosts connected by local or wide area network. Each client application communicates directly with the hosts that serve data used by the application. Any data transfer takes place between the client process and the Objectivity page server (ooams) which runs on each data-serving host\(^2\). Consistency for concurrent access is provided through one or more lockserver processes per federation. Before any data is read or modified, the client connects to this lockserver to obtain a suitable lock.

![Diagram of Distributed Applications Sharing a Federated Database](image)

**Figure 6: Distributed Applications Sharing a Federated Database**

3.2 Data Replication

Objectivity/DB allows replicating all objects in a particular database to multiple physical locations. The aim of this data replication is twofold:

- **Enhance performance:**
  Client programs may access a local copy of the data instead of transferring data over a network.
- **Enhance availability:**
  Clients on sites which are temporarily disconnected from the full data store may continue to work on the subset of data for which local replicas are available.

The following diagram shows a simple configuration where one database is replicated from site 1 to two other remote sites over a wide area network.

---

\(^2\) Processes that run on the same host on which the database is located directly access the database files without using the page server.
Any state changes of replicated objects on either site are transparently propagated to all other replicas by the database system. In the case that some of the replicas are not reachable, a quorum-based mechanism is used to determine which replica may be modified and a backlog of all changes is kept until other replicas become online again.

The data replication feature is expected to be very useful, for example to distribute central event selection data to multiple regional data centers.

### 3.3 Schema Evolution

Given the extremely long time scale for LHC experiments it is important to foresee the possibility to change the object model of experiment data during the experiment lifetime. Not only new persistent classes need to be incorporated into the federation schema but also existing class definitions will need to be changed.

The schema evolution feature of Objectivity/DB allows for example to add, move or remove attributes within classes or to change the inheritance hierarchy between persistent classes. If a schema change affects any existing persistent objects the database provides a flexible migration scheme. Depending for example on the amount of involved data one can choose between:

- Immediate Conversion
  - all affected objects are converted to the new class layout using an upgrade application
- Lazy Conversion:
  - affected objects are upgraded as they are accessed

### 3.4 Object Versioning

Several ODBMS systems provide support for maintaining multiple versions of the same logical object. In Objectivity these versions may for example constitute a simple linear time series of states of one object or form a more complex tree of states. The following diagram shows a general version genealogy involving multiple linear versions and branches of one object.
The object versioning feature is used to implement the calibration database of the BaBar experiment.

### 3.5 Other ODBMS Products

**Versant**

Versant uses a different splitting between database client and server than Objectivity/DB. Whereas Objectivity/DB implements a "fat-client/thin-server" model, Versant provides the opposite. The object identifier (OID) in Objectivity/DB has a nearly direct physical mapping, whereas Versant uses a logical object identifier (LOID).

- In Versant, distributed databases are less tightly coupled than in Objectivity/DB. Each database has its own schema, as opposed to Objectivity/DB, where the schema is shared across the entire federation. In our environment, where many databases are likely to reside offline on tape schema inconsistencies between different databases are likely to happen.
- Although Versant implements a LOID, an application must know in which database each object was created and is responsible for opening the databases in question. This contrasts strongly with Objectivity/DB where it is sufficient to initialise access to the federation.
- The support for object clustering in Versant is less flexible than in Objectivity/DB.

### 4 HEP PRODUCTION SCENARIOS: ODBMS-BASED DATA ANALYSIS

A typical analysis scenario can be split in two parts. The first part concerns populating the database with event data and is usually done in a non-interactive C++ program (e.g., in batch mode). The second part implies using an interactive tool, such as the IRIS Explorer framework, to actually produce summary data, usually as histograms, out of the event data.

**4.1.1 Building a Tag Database**

In this first stage, the analysis data store, assumed to be provided by Objectivity/DB and HPSS, is populated, e.g. from a former reconstruction phase. The following figure shows schematically the difference between the PAW+Ntuple and tagDB models.
Rather than imposing a pre-defined data model, as is the case with today's NTuple, the LHC++ analysis tools work directly with the data model of the experiment. To assist in the selection of the required events, a new concept is introduced - that of an event tag. An event tag is a small object containing selection relevant attributes for each event. The tag objects are stored physically separate from the event to permit efficient clustering, but have an association to the full event data. At the moment, two classes of tags have been implemented:

- **Concrete tags** have their own schema, and are recommended for experiment-wide or work-group activities, as they offer optimal performance.

- **Generic tags**, on the other hand, are more suited for individual physicists - they can be defined on the fly and do not require the definition of new schema, albeit with a small, but acceptable, performance penalty.

Both implementations share the same interface, which is entirely decoupled from the physical storage model. This permits the implementation of different clustering strategies, such as attribute-based clustering - as in column-wise NTuple - without affecting the user interface.

```cpp
// create a new tag collection
GenericTag simTag("simulation tag");

// define all attributes of my tags
TagAttribute<long>  evtNo(simTag,"event number");
TagAttribute<float> et   (simTag,"Et particle1");
TagAttribute<float> theta(simTag,"theta particle1");
TagAttribute<short> pid  (simTag,"id particle1");
```

**Example 2 - Creation and Definition of a New Event Tag**

Tags may be filled in a simple event loop, as shown below. It is important to note that the tag attributes are handled just like normal C++ variables.
while ( evt = geant->nextEvent() )
{
    simTag.newTag(); // create a new tag
    et = evt->getPart(1).et;
    theta = evt->getPart(1).theta;
    pid = evt->getPart(1).pdg_code;
}

Example 3 - Filling a Previously Defined Tag

As has been described above, a fundamental feature of this strategy is the ease in which the full event data can be accessed. This is a significant piece of new functionality that was not possible using PAW and NTuples.

while (atlasTags->next())
{
    if (et > 4.5 && sin(theta) > .5) // for selected events…
        // … fill histograms from the tag…
        cout << "event: " << eventNo << endl;
        etHisto->fill(et);
        thetaHisto->fill(theta);
    // … but also using data from the event.
    ntracks = atlasTags->event->tracking->trackList.size();
    nTracksHisto->fill(nTracks);
}

Example 4 - Accessing the Event Data from the Tag

Having populated a collection of tags - typically, but not necessarily, performed in batch, these data can then be directly be visualised e.g. using IRIS Explorer.

4.1.2 Interactive Data Analysis

Having defined and populated collections of tags, these can then be analysed in the IRIS Explorer framework, using a combination of standard and HEP-specific modules. A user does not need to be exposed to the details - a collection of modules can be predefined and presented to the user as a complete application. Functionality similar to that provided in PAW is available, but with a number of significant differences. The most important of these is that analysis is no longer restricted to the subset of data that has been copied into a NTuple. A user has access to all of the data for a given event and is presented with a unified interface to the entire data store. This includes not only the event data, but also associated meta-data, such as calibration information, the definitions of various event selections, and so on. In this way, is it possible to build associations from histograms to the selection criteria that produced them and the data itself.

In the past, the user was restricted to that subset of the data that was copied into a given NTuple. Furthermore, the management of these NTuple was entirely the responsibility of the user (naming, storing etc.). In the ODBMS approach, the user has transparent access to any event or indeed any part of the event that is stored in a consistent manner experiment wide. These features offer considerable improvements in flexibility and reproducibility over previous approaches.
5 HEP PROJECTS BASED ON OBJECTIVITY/DB

5.1.1 Babar
The Babar experiment at SLAC is expected to start taking data in 1999. Babar uses Objectivity to store event, simulation, calibration and analysis data of an expected amount of 200TB/year. The majority of this storage will be managed by HPSS.

5.1.2 ZEUS
ZEUS is a large detector electron-proton collider HERA at DESY. Since ZEUS started taking data already in 1992, the analysis environment at ZEUS is mainly based FORTRAN using the ADAMO system. Since 1996 Objectivity is used for event selection in the analysis phase. About 20GB of “tag data” are used with the plan to extend to a store of 200GB. The result of re-implementing the event selection using Objectivity/DB was a significant gain in performance and flexibility compared to the old system.

5.1.3 AMS
The Alpha Magnetic Spectrometer (AMS) took its first data on a NASA space shuttle flight in 1998 and will be used later on the International Space Station. AMS’s main research goal is the search for antimatter and dark matter. During its first data-taking period Objectivity was mainly used to store production data, slow control parameters and NASA auxiliary data.

5.1.4 CERES
The CERES/NA45 experiment is a heavy ion experiment at the CERN SPS studying electron-positron pairs in relativistic nuclear collisions. In 1997, CERES successfully used Objectivity/DB to perform a parallel reconstruction and filtering of their raw data from a multiprocessor farm (32 nodes MEIKO/Quadrics CS2). After recently upgrading their detector with a time projection chamber, CERES expects in 1999 to write 30 TB of raw data during a data-taking period of 1 month.

5.1.5 CHORUS
The CHORUS experiment searching for neutrino oscillations uses Objectivity/DB for an online emulsion scan database. CHORUS plans to deploy this application also at sites outside CERN.

5.1.6 COMPASS
The COMPASS experiment expects to begin full data taking in 2000 with a preliminary run in 1999. Some 300TB of raw data will be acquired per year at rates up to 35MB/second. Analysis data is expected to be stored on disk, requiring some 3-20TB of disk space. Some 50 concurrent users and many passes through the data are expected.

6 CONCLUSION
HEP data stores based on Object Database Management Systems (ODBMS) provide a number of important advantages in comparison with traditional systems. The database approach provides the user with in a coherent logical view of complex HEP object models and allows a tight integration with multiple of today’s OO languages such as C++ and JAVAX.

The clear separation of logical and physical data model introduced by object databases allows for transparent support of physical clustering and re-clustering of data which is expected to be an important tool to optimise the overall system performance.

The ODBMS implementation of Objectivity/DB in particular show scaling up to multi-PB distributed data stores and provides a seamless integration with Mass Storage Systems (MSS) like HPSS. Already today a significant number of HEP experiments in or close to production adopted an ODBMS based approach.
BIBLIOGRAPHY

RD45 - A Persistent Object Manager for HEP, LCB Status Report, March 1998, CERN/LHCC 98-x
Using an Object Database and Mass Storage System for Physics Production, March 1998, CERN/LHCC 98-x
RD45 - A Persistent Object Manager for HEP, LCB Status Report, March 1997, CERN/LHCC 97-6
Object Databases and their Impact on Storage-Related Aspects of HEP Computing, the RD45 collaboration
Object Database Features and HEP Data Management, the RD45 collaboration
Using and Object Database and Mass Storage System for Physics Analysis, the RD45 collaboration
RD45 - A Persistent Object Manager for HEP, LCRB Status Report, March 1996, CERN/LHCC 96-15
Object Databases and Mass Storage Systems: The Prognosis, the RD45 collaboration, CERN/LHCC 96-17
DBMS Needs Assessment for Objects, Barry and Associates (release 3)
ATLAS Computing Technical Proposal, CERN/LHCC 96-43
CMS Computing Technical Proposal, CERN/LHCC 96-45
Abstract
This article introduces the key concepts underlying computational grids. The article is structured in terms of six questions: Why do we need computational grids? What types of applications will they be used for? Who will use grids? How will they program them? What is involved in building a grid? What problems remain to be solved before grids become commonplace?

1. INTRODUCTION
This introductory chapter lays the groundwork for the rest of the book by providing a more detailed picture of the expected purpose, shape, and architecture of future grid systems. We structure the chapter in terms of six questions that we believe are central to this discussion: Why do we need computational grids? What types of applications will grids be used for? Who will use grids? How will grids be used? What is involved in building a grid? And, what problems must be solved to make grids commonplace?
We provide an overview of each of these issues here, referring to subsequent chapters for more detailed discussion.

2. REASONS FOR COMPUTATIONAL GRIDS
Why do we need computational grids? Computational approaches to problem solving have proven their worth in almost every field of human endeavor. Computers are used for modeling and simulating complex scientific and engineering problems, diagnosing medical conditions, controlling industrial equipment, forecasting the weather, managing stock portfolios, and many other purposes. Yet, although there are certainly challenging problems that exceed our ability to solve them, computers are still used much less extensively than they could be. To pick just one example, university researchers make extensive use of computers when studying the impact of changes in land use on biodiversity, but city planners selecting routes for new roads or planning new zoning ordinances do not. Yet it is local decisions such as these that, ultimately, shape our future.

There are a variety of reasons for this relative lack of use of computational problem-solving methods, including lack of appropriate education and tools. But one important factor is that the average computing environment remains inadequate for such computationally sophisticated purposes. While today’s PC is faster than the Cray supercomputer of 10 years ago, it is still far from adequate for predicting the outcome of complex actions or selecting from among many choices. That, after all, is why supercomputers have continued to evolve.

2.1 Increasing delivered computation
We believe that the opportunity exists to provide users—whether city planners, engineers, or scientists—with substantially more computational power: an increase of three orders of magnitude within five years, and five orders of magnitude within a decade. These dramatic increases will be achieved by innovations in a wide range of areas:

1. **Technology improvement**: Evolutionary changes in VLSI technology and microprocessor architecture can be expected to result in a factor of 10 increase in computational capabilities in the next five years, and a factor of 100 increase in the next ten.

2. **Increase in demand-driven access to computational power**: Many applications have only episodic requirements for substantial computational resources. For example, a medical diagnosis system may be run only when a cardiogram is performed, a stockmarket simulation only when a user recomputes retirement benefits, or a seismic simulation only after a major earthquake. If mechanisms are in place to allow reliable, instantaneous, and transparent access to high-end resources, then from the perspective of these applications it is as if those resources are dedicated to them. Given the existence of multiteraFLOPS systems, an increase in apparent computational power of three or more orders of magnitude is feasible.

3. **Increased utilization of idle capacity**: Most low-end computers (PCs and workstations) are often idle: various studies report utilizations of around 30% in academic and commercial environments [47], [21]. Utilization can be increased by a factor of two, even for parallel programs [4], without impinging significantly on productivity. The benefit to individual users can be substantially greater: factors of 100 or 1,000 increase in peak computational capacity have been reported [41], [75].

4. **Greater sharing of computational results**: The daily weather forecast involves perhaps $10^{14}$ numerical operations. If we assume that the forecast is of benefit to $10^7$ people, we have $10^{21}$ effective operations—comparable to the computation performed each day on all the world’s PCs. Few other computational results or facilities are shared so effectively today, but they may be in the future as other scientific communities adopt a “big science” approach to computation. The key to more sharing may be the development of collaboratories: “...center[s] without walls, in which the nation’s researchers can perform their research without regard to geographical location—interacting with colleagues, accessing instrumentation, sharing data and computational resources, and accessing information in digital libraries” [48].

5. **New problem-solving techniques and tools**: A variety of approaches can improve the efficiency or ease with which computation is applied to problem solving. For example, network-enabled solvers [17], [11] allow users to invoke advanced numerical solution methods without having to install sophisticated software. Teleimmersion techniques [50] facilitate the sharing of computational results by supporting collaborative steering of simulations and exploration of data sets.

Underlying each of these advances is the synergistic use of high-performance networking, computing, and advanced software to provide access to advanced computational capabilities, regardless of the location of users and resources.

### 2.2 Definition of computational grids

The current status of computation is analogous in some respects to that of electricity around 1910. At that time, electric power generation was possible, and new devices were being devised that depended on electric power, but the need for each user to build and operate a new generator hindered use. The truly revolutionary development was not, in fact, electricity, but the electric power grid and the associated transmission and distribution technologies. Together, these developments provided reliable, low-cost access to a standardized service, with the result that power—which for most of human history has been accessible only in crude and not especially portable forms (human effort, horses, water power, steam engines, candles)—became universally accessible. By allowing both individuals and industries to take for granted the availability of cheap, reliable power, the electric power grid made possible both new devices and the new industries that manufactured them.

By analogy, we adopt the term *computational grid* for the infrastructure that will enable the increases in computation discussed above. A computational grid is a hardware and software infrastructure
that provides dependable, consistent, pervasive, and inexpensive access to high-end computational capabilities.

We talk about an infrastructure because a computational grid is concerned, above all, with large-scale pooling of resources, whether compute cycles, data, sensors, or people. Such pooling requires significant hardware infrastructure to achieve the necessary interconnections and software infrastructure to monitor and control the resulting ensemble. In the rest of this chapter, and throughout the book, we discuss in detail the nature of this infrastructure.

The need for dependable service is fundamental. Users require assurances that they will receive predictable, sustained, and often high levels of performance from the diverse components that constitute the grid; in the absence of such assurances, applications will not be written or used. The performance characteristics that are of interest will vary widely from application to application, but may include network bandwidth, latency, jitter, computer power, software services, security, and reliability.

The need for consistency of service is a second fundamental concern. As with electric power, we need standard services, accessible via standard interfaces, and operating within standard parameters. Without such standards, application development and pervasive use are impractical. A significant challenge when developing standards is to encapsulate heterogeneity without compromising high-performance execution.

Pervasive access allows us to count on services always being available, within whatever environment we expect to move. Pervasiveness does not imply that resources are everywhere or are universally accessible. We cannot access electric power in a new home until wire has been laid and an account established with the local utility; computational grids will have similarly circumscribed availability and controlled access. However, we will be able to count on universal access within the confines of whatever environment the grid is designed to support.

Finally, an infrastructure must offer inexpensive (relative to income) access if it is to be broadly accepted and used. Homeowners and industrialists both make use of remote billion-dollar power plants on a daily basis because the cost to them is reasonable. A computational grid must achieve similarly attractive economics.

It is the combination of dependability, consistency, and pervasiveness that will cause computational grids to have a transforming effect on how computation is performed and used. By increasing the set of capabilities that can be taken for granted to the extent that they are noticed only by their absence, grids allow new tools to be developed and widely deployed. Much as pervasive access to bitmapped displays changed our baseline assumptions for the design of application interfaces, computational grids can fundamentally change the way we think about computation and resources.

2.3 The impact of grids

The history of network computing shows that orders-of-magnitude improvements in underlying technology invariably enable revolutionary, often unanticipated, applications of that technology, which in turn motivate further technological improvements. As a result, our view of network computing has undergone repeated transformations over the past 40 years.

There is considerable evidence that another such revolution is imminent. The capabilities of both computers and networks continue to increase dramatically. Ten years of research on metacomputing has created a solid base of experience in new applications that couple high-speed networking and computing. The time seems ripe for a transition from the heroic days of metacomputing to more integrated computational grids with dependable and pervasive computational capabilities and consistent interfaces. In such grids, today’s metacomputing applications will be routine, and programmers will be able to explore a new generation of yet more interesting applications that leverage teraFLOP computers and petabyte storage systems interconnected by gigabit networks. We present two simple examples to illustrate how grid functionality may transform different aspects of our lives.
Today’s home finance software packages leverage the pervasive availability of communication technologies such as modems, Internet service providers, and the Web to integrate up-to-date stock prices obtained from remote services into local portfolio value calculations. However, the actual computations performed on this data are relatively simple. In tomorrow’s grid environment, we can imagine individuals making stock-purchasing decisions on the basis of detailed Monte Carlo analyses of future asset value, performed on remote teraFLOP computers. The instantaneous use of three orders of magnitude more computing power than today will go unnoticed by prospective retirees, but their lives will be different because of more accurate information.

Today, citizen groups evaluating a proposed new urban development must study uninspiring blueprints or perspective drawings at city hall. A computational grid will allow them to call on powerful graphics computers and databases to transform the architect’s plans into realistic virtual reality depictions and to explore such design issues as energy consumption, lighting efficiency, or sound quality. Meeting online to walk through and discuss the impact of the new development on their community, they can arrive at better urban design and hence improved quality of life. Virtual reality-based simulation models of Los Angeles, produced by William Jepson, and the walkthrough model of Soda Hall at the University of California–Berkeley, constructed by Carlo Seguin and his colleagues, are interesting exemplars of this use of computing [9].

2.4 Electric power grids
We conclude this section by reviewing briefly some salient features of the computational grid’s namesake. The electric power grid is remarkable in terms of its construction and function, which together make it one of the technological marvels of the 20th century. Within large geographical regions (e.g., North America), it forms essentially a single entity that provides power to billions of devices, in a relatively efficient, low-cost, and reliable fashion. The North American grid alone links more than ten thousand generators with billions of outlets via a complex web of physical connections and trading mechanisms [12]. The components from which the grid is constructed are highly heterogeneous in terms of their physical characteristics and are owned and operated by different organizations. Consumers differ significantly in terms of the amount of power they consume, the service guarantees they require, and the amount they are prepared to pay.

Analogies are dangerous things, and electricity is certainly very different from computation in many respects. Nevertheless, the following aspects of the power grid seem particularly relevant to the current discussion.

Importance of economics
The role and structure of the power grid are driven to a large extent by economic factors. Oil- and coal-fired generators have significant economies of scale. A power company must be able to call upon reserve capacity equal to its largest generator in case that generator fails; interconnections between regions allow for sharing of such reserve capacity, as well as enabling trading of excess power. The impact of economic factors on computational grids is not well understood [34]. Where and when are there economies of scale to be obtained in computational capabilities? Might economic factors lead us away from today’s model of a “computer on every desktop”? We note an intriguing development. Recent advances in power generation technology (e.g., small gas turbines) and the deregulation of the power industry are leading some analysts to look to the Internet for lessons regarding the future evolution of the electric power grid!

Importance of politics
The developers of large-scale grids tell us that their success depended on regulatory, political, and institutional developments as much as on technical innovation [12]. This lesson should be taken to heart by developers of future computational grids.
Complexity of control

The principal technical challenges in power grids—once technology issues relating to efficient generation and high-voltage transmission had been overcome—relate to the management of a complex ensemble in which changes at a single location can have far-reaching consequences [12]. Hence, we find that the power grid includes a sophisticated infrastructure for monitoring, management, and control. Again, there appear to be many parallels between this control problem and the problem of providing performance guarantees in large-scale, dynamic, and heterogeneous computational grid environments.

3. GRID APPLICATIONS

What types of applications will grids be used for? Building on experiences in gigabit testbeds [42], [59], the I-WAY network [19], and other experimental systems, we have identified five major application classes for computational grids, listed in Table 1 and described briefly in this section. More details about applications and their technical requirements are provided in the referenced chapters.

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
<th>Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed supercomputing</td>
<td>DIS Stellar dynamics Ab initio chemistry</td>
<td>Very large problems needing lots of CPU, memory, etc.</td>
</tr>
<tr>
<td>High throughput</td>
<td>Chip design Parameter studies Cryptographic problems</td>
<td>Harness many otherwise idle resources to increase aggregate throughput</td>
</tr>
<tr>
<td>On demand</td>
<td>Medical instrumentation Network-enabled solvers Cloud detection</td>
<td>Remote resources integrated with local computation, often for bounded amount of time</td>
</tr>
<tr>
<td>Data intensive</td>
<td>Sky survey Physics data Data assimilation</td>
<td>Synthesis of new information from many or large data sources</td>
</tr>
<tr>
<td>Collaborative</td>
<td>Collaborative design Data exploration Education</td>
<td>Support communication or collaborative work between multiple participants</td>
</tr>
</tbody>
</table>

3.1 Distributed supercomputing

Distributed supercomputing applications use grids to aggregate substantial computational resources in order to tackle problems that cannot be solved on a single system. Depending on the grid on which we are working (see Section 4.), these aggregated resources might comprise the majority of the supercomputers in the country or simply all of the workstations within a company. Here are some contemporary examples:

- Distributed interactive simulation (DIS) is a technique used for training and planning in the military. Realistic scenarios may involve hundreds of thousands of entities, each with potentially complex behavior patterns. Yet even the largest current supercomputers can handle at most 20,000 entities. In recent work, researchers at the California Institute of Technology have shown how multiple supercomputers can be coupled to achieve record-breaking levels of performance.
- The accurate simulation of complex physical processes can require high spatial and temporal resolution in order to resolve fine-scale detail. Coupled supercomputers can be used in such situations to overcome resolution barriers and hence to obtain qualitatively new scientific results. Although high latencies can pose significant obstacles, coupled supercomputers have been used success-
fully in cosmology [54], high-resolution ab initio computational chemistry computations [52], and climate modeling [45].

Challenging issues from a grid architecture perspective include the need to coschedule what are often scarce and expensive resources, the scalability of protocols and algorithms to tens or hundreds of thousands of nodes, latency-tolerant algorithms, and achieving and maintaining high levels of performance across heterogeneous systems.

3.2 High-throughput computing

In high-throughput computing, the grid is used to schedule large numbers of loosely coupled or independent tasks, with the goal of putting unused processor cycles (often from idle workstations) to work. The result may be, as in distributed supercomputing, the focusing of available resources on a single problem, but the quasi-independent nature of the tasks involved leads to very different types of problems and problem-solving methods. Here are some examples:

- Platform Computing Corporation reports that the microprocessor manufacturer Advanced Micro Devices used high-throughput computing techniques to exploit over a thousand computers during the peak design phases of their K6 and K7 microprocessors. These computers are located on the desktops of AMD engineers at a number of AMD sites and were used for design verification only when not in use by engineers.
- The Condor system from the University of Wisconsin is used to manage pools of hundreds of workstations at universities and laboratories around the world [41]. These resources have been used for studies as diverse as molecular simulations of liquid crystals, studies of ground-penetrating radar, and the design of diesel engines.
- More loosely organized efforts have harnessed tens of thousands of computers distributed worldwide to tackle hard cryptographic problems [40].

3.3 On-demand computing

On-demand applications use grid capabilities to meet short-term requirements for resources that cannot be cost-effectively or conveniently located locally. These resources may be computation, software, data repositories, specialized sensors, and so on. In contrast to distributed supercomputing applications, these applications are often driven by cost-performance concerns rather than absolute performance. For example:

- The NEOS [17] and NetSolve [11] network-enhanced numerical solver systems allow users to couple remote software and resources into desktop applications, dispatching to remote servers calculations that are computationally demanding or that require specialized software.
- A computer-enhanced MRI machine and scanning tunneling microscope (STM) developed at the National Center for Supercomputing Applications use supercomputers to achieve real-time image processing [57], [58]. The result is a significant enhancement in the ability to understand what we are seeing and, in the case of the microscope, to steer the instrument.
- A system developed at the Aerospace Corporation for processing of data from meteorological satellites uses dynamically acquired supercomputer resources to deliver the results of a cloud detection algorithm to remote meteorologists in quasi real time [38].

The challenging issues in on-demand applications derive primarily from the dynamic nature of resource requirements and the potentially large populations of users and resources. These issues include resource location, scheduling, code management, configuration, fault tolerance, security, and payment mechanisms.
3.4 Data-intensive computing

In data-intensive applications, the focus is on synthesizing new information from data that is maintained in geographically distributed repositories, digital libraries, and databases. This synthesis process is often computationally and communication intensive as well.

- Future high-energy physics experiments will generate terabytes of data per day, or around a petabyte per year. The complex queries used to detect “interesting” events may need to access large fractions of this data [43]. The scientific collaborators who will access this data are widely distributed, and hence the data systems in which data is placed are likely to be distributed as well.
- The Digital Sky Survey will, ultimately, make many terabytes of astronomical photographic data available in numerous network-accessible databases. This facility enables new approaches to astronomical research based on distributed analysis, assuming that appropriate computational grid facilities exist.
- Modern meteorological forecasting systems make extensive use of data assimilation to incorporate remote satellite observations. The complete process involves the movement and processing of many gigabytes of data.

Challenging issues in data-intensive applications are the scheduling and configuration of complex, high-volume data flows through multiple levels of hierarchy.

3.5 Collaborative computing

Collaborative applications are concerned primarily with enabling and enhancing human-to-human interactions. Such applications are often structured in terms of a virtual shared space. Many collaborative applications are concerned with enabling the shared use of computational resources such as data archives and simulations; in this case, they also have characteristics of the other application classes just described. For example:

- The BoilerMaker system developed at Argonne National Laboratory allows multiple users to collaborate on the design of emission control systems in industrial incinerators [20]. The different users interact with each other and with a simulation of the incinerator.
- The CAVE5D system supports remote, collaborative exploration of large geophysical data sets and the models that generate them—for example, a coupled physical/biological model of the Chesapeake Bay [74].
- The NICE system developed at the University of Illinois at Chicago allows children to participate in the creation and maintenance of realistic virtual worlds, for entertainment and education [60].

Challenging aspects of collaborative applications from a grid architecture perspective are the real-time requirements imposed by human perceptual capabilities and the rich variety of interactions that can take place.

We conclude this section with three general observations. First, we note that even in this brief survey we see a tremendous variety of already successful applications. This rich set has been developed despite the significant difficulties faced by programmers developing grid applications in the absence of a mature grid infrastructure. As grids evolve, we expect the range and sophistication of applications to increase dramatically. Second, we observe that almost all of the applications demonstrate a tremendous appetite for computational resources (CPU, memory, disk, etc.) that cannot be met in a timely fashion by expected growth in single-system performance. This emphasizes the importance of grid technologies as a means of sharing computation as well as a data access and communication medium. Third, we see that many of the applications are interactive, or depend on tight synchronization with computational components, and hence depend on the availability of a grid infrastructure able to provide robust performance guarantees.
4. GRID COMMUNITIES

Who will use grids? One approach to understanding computational grids is to consider the communities that they serve. Because grids are above all a mechanism for sharing resources, we ask, What groups of people will have sufficient incentive to invest in the infrastructure required to enable sharing, and what resources will these communities want to share?

One perspective on these questions holds that the benefits of sharing will almost always outweigh the costs and, hence, that we will see grids that link large communities with few common interests, within which resource sharing will extend to individual PCs and workstations. If we compare a computational grid to an electric power grid, then in this view, the grid is quasi-universal, and every user has the potential to act as a cogenerator. Skeptics respond that the technical and political costs of sharing resources will rarely outweigh the benefits, especially when coupling must cross institutional boundaries. Hence, they argue that resources will be shared only when there is considerable incentive to do so: because the resource is expensive, or scarce, or because sharing enables human interactions that are otherwise difficult to achieve. In this view, grids will be specialized, designed to support specific user communities with specific goals.

Rather than take a particular position on how grids will evolve, we propose what we see as four plausible scenarios, each serving a different community. Future grids will probably include elements of all four.

4.1 Government

The first community that we consider comprises the relatively small number—thousands or perhaps tens of thousands—of officials, planners, and scientists concerned with problems traditionally assigned to national government, such as disaster response, national defense, and long-term research and planning. There can be significant advantage to applying the collective power of the nation's fastest computers, data archives, and intellect to the solution of these problems. Hence, we envision a grid that uses the fastest networks to couple relatively small numbers of high-end resources across the nation—perhaps tens of teraFLOP computers, petabytes of storage, hundreds of sites, thousands of smaller systems—for two principal purposes:

1. To provide a “strategic computing reserve,” allowing substantial computing resources to be applied to large problems in times of crisis, such as to plan responses to a major environmental disaster, earthquake, or terrorist attack
2. To act as a “national collaboratory,” supporting collaborative investigations of complex scientific and engineering problems, such as global change, space station design, and environmental cleanup

An important secondary benefit of this high-end national supercomputing grid is to support resource trading between the various operators of high-end resources, hence increasing the efficiency with which those resources are used.

This national grid is distinguished by its need to integrate diverse high-end (and hence complex) resources, the strategic importance of its overall mission, and the diversity of competing interests that must be balanced when allocating resources.

4.2 A health maintenance organization

In our second example, the community supported by the grid comprises administrators and medical personnel located at a small number of hospitals within a metropolitan area. The resources to be shared are a small number of high-end computers, hundreds of workstations, administrative databases, medical image archives, and specialized instruments such as MRI machines, CAT scanners, and cardioangiography devices. The coupling of these resources into an integrated grid enables a wide range of new, computationally enhanced applications: desktop tools that use centralized supercomputer resources to run
computer-aided diagnosis procedures on mammograms or to search centralized medical image archives for similar cases; life-critical applications such as telerobotic surgery and remote cardiac monitoring and analysis; auditing software that uses the many workstations across the hospital to run fraud detection algorithms on financial records; and research software that uses supercomputers and idle workstations for epidemiological research. Each of these applications exists today in research laboratories, but has rarely been deployed in ordinary hospitals because of the high cost of computation.

This private grid is distinguished by its relatively small scale, central management, and common purpose on the one hand, and on the other hand by the complexity inherent in using common infrastructure for both life-critical applications and less reliability-sensitive purposes and by the need to integrate low-cost commodity technologies. We can expect grids with similar characteristics to be useful in many institutions.

4.3 A materials science collaboratory

The community in our third example is a group of scientists who operate and use a variety of instruments, such as electron microscopes, particle accelerators, and X-ray sources, for the characterization of materials. This community is fluid and highly distributed, comprising many hundreds of university researchers and students from around the world, in addition to the operators of the various instruments (tens of instruments, at perhaps ten centers). The resources that are being shared include the instruments themselves, data archives containing the collective knowledge of this community, sophisticated analysis software developed by different groups, and various supercomputers used for analysis. Applications enabled by this grid include remote operation of instruments, collaborative analysis, and supercomputer-based online analysis.

This virtual grid is characterized by a strong unifying focus and relatively narrow goals on the one hand, and on the other hand by dynamic membership, a lack of central control, and a frequent need to coexist with other uses of the same resources. We can imagine similar grids arising to meet the needs of a variety of multi-institutional research groups and multicompany virtual teams created to pursue long- or short-term goals.

4.4 Computational market economy

The fourth community that we consider comprises the participants in a broad-based market economy for computational services. This is a potentially enormous community with no connections beyond the usual market-oriented relationships. We can expect participants to include consumers, with their diverse needs and interests; providers of specialized services, such as financial modeling, graphics rendering, and interactive gaming; providers of compute resources; network providers, who contract to provide certain levels of network service; and various other entities such as banks and licensing organizations.

This public grid is in some respects the most intriguing of the four scenarios considered here, but is also the least concrete. One area of uncertainty concerns the extent to which the average consumer will also act as a producer of computational resources. The answer to this question seems to depend on two issues. Will applications emerge that can exploit loosely coupled computational resources? And, will owners of resources be motivated to contribute resources? To date, large-scale activity in this area has been limited to fairly esoteric computations—such as searching for prime numbers, breaking cryptographic codes [40], or detecting extraterrestrial communications [64]—with the benefit to the individuals being the fun of participating and the potential momentary fame if their computer solves the problem in question.

We conclude this section by noting that, in our view, each of these scenarios seems quite feasible; indeed, substantial prototypes have been created for each of the grids that we describe. Hence, we expect to see not just one single computational grid, but rather many grids, each serving a different community with its own requirements and objectives. Just which grids will evolve depends critically on three issues:
the evolving economics of computing and networking, and the services that these physical infrastructure elements are used to provide; the institutional, regulatory, and political frameworks within which grids may develop; and, above all, the emergence of applications able to motivate users to invest in and use grid technologies.

5. USING GRIDS

How will grids be used? In metacomputing experiments conducted to date, users have been “heroic” programmers, willing to spend large amounts of time programming complex systems at a low level. The resulting applications have provided compelling demonstrations of what might be, but in most cases are too expensive, unreliable, insecure, and fragile to be considered suitable for general use.

For grids to become truly useful, we need to take a significant step forward in grid programming, moving from the equivalent of assembly language to high-level languages, from one-off libraries to application toolkits, and from hand-crafted codes to shrink-wrapped applications. These goals are familiar to us from conventional programming, but in a grid environment we are faced with the additional difficulties associated with wide area operation—in particular, the need for grid applications to adapt to changes in resource properties in order to meet performance requirements. As in conventional computing, an important step toward the realization of these goals is the development of standards for applications, programming models, tools, and services, so that a division of labor can be achieved between the users and developers of different types of components.

We structure our discussion of grid tools and programming in terms of the classification illustrated in Table 2. At the lowest level, we have grid developers—the designers and implementors of what we might call the “Grid Protocol,” by analogy with the Internet Protocol that provides the lowest-level services in the Internet—who provide the basic services required to construct a grid. Above this, we have tool developers, who use grid services to construct programming models and associated tools, layering higher-level services and abstractions on top of the more fundamental services provided by the grid architecture. Application developers, in turn, build on these programming models, tools, and services to construct grid-enabled applications for end users who, ideally, can use these applications without being concerned with the fact that they are operating in a grid environment. A fifth class of users, system administrators, is responsible for managing grid components. We now examine this model in more detail.

<table>
<thead>
<tr>
<th>Class</th>
<th>Purpose</th>
<th>Makes use of</th>
<th>Concerns</th>
</tr>
</thead>
<tbody>
<tr>
<td>End users</td>
<td>Solve problems</td>
<td>Applications</td>
<td>Transparency, performance</td>
</tr>
<tr>
<td>Application developers</td>
<td>Develop applications</td>
<td>Programming models, tools</td>
<td>Ease of use, performance</td>
</tr>
<tr>
<td>Tool developers</td>
<td>Develop tools, programming models</td>
<td>Grid services</td>
<td>Adaptivity, exposure of performance, security</td>
</tr>
<tr>
<td>Grid developers</td>
<td>Provide basic grid services</td>
<td>Local system services</td>
<td>Local simplicity, connectivity, security</td>
</tr>
<tr>
<td>System administrators</td>
<td>Manage grid resources</td>
<td>Management tools</td>
<td>Balancing local and global concerns</td>
</tr>
</tbody>
</table>

5.1 Grid developers

A very small group of grid developers are responsible for implementing the basic services referred to above. We discuss the concerns encountered at this level in Section 6.
5.2 Tool developers

Our second group of users are the developers of the tools, compilers, libraries, and so on that implement the programming models and services used by application developers. Today’s small population of grid tool developers (e.g., the developers of Condor [41], Nimrod [1], NEOS [17], NetSolve [11], Horus [68], grid-enabled implementations of the Message Passing Interface (MPI) [27], and CAVERN [39]) must build their tools on a very narrow foundation, comprising little more than the Internet Protocol. We envision that future grid systems will provide a richer set of basic services, hence making it possible to build more sophisticated and robust tools. We discuss the nature and implementation of those basic services in Section 6.; briefly, they comprise versions of those services that have proven effective on today’s end systems and clusters, such as authentication, process management, data access, and communication, plus new services that address specific concerns of the grid environment, such as resource location, information, fault detection, security, and electronic payment.

Tool developers must use these basic services to provide efficient implementations of the programming models that will be used by application developers. In constructing these translations, the tool developer must be concerned not only with translating the existing model to the grid environment, but also with revealing to the programmer those aspects of the grid environment that impact performance. For example, a grid-enabled MPI [27] can seek to adapt the MPI model for grid execution by incorporating specialized techniques for point-to-point and collective communication in highly heterogeneous environments; implementations of collective operations might use multicast protocols and adapt a combining tree structure in response to changing network loads. It should probably also extend the MPI model to provide programmers with access to resource location services, information about grid topology, and group communication protocols.

5.3 Application developers

Our third class of users comprises those who construct grid-enabled applications and components. Today, these programmers write applications in what is, in effect, an assembly language: explicit calls to the Internet Protocol’s User Datagram Protocol (UDP) or Transmission Control Protocol (TCP), explicit or no management of failure, hard-coded configuration decisions for specific computing systems, and so on. We are far removed from the portable, efficient, high-level languages that are used to develop sequential programs, and the advanced services that programmers can rely upon when using these languages, such as dynamic memory management and high-level I/O libraries.

Future grids will need to address the needs of application developers in two ways. They must provide programming models (supported by languages, libraries, and tools) that are appropriate for grid environments and a range of services (for security, fault detection, resource management, data access, communication, etc.) that programmers can call upon when developing applications.

The purpose of both programming models and services is to simplify thinking about and implementing complex algorithmic structures, by providing a set of abstractions that hide details unrelated to the application, while exposing design decisions that have a significant impact on program performance or correctness. In sequential programming, commonly used programming models provide us with abstractions such as subroutines and scoping; in parallel programming, we have threads and condition variables (in shared-memory parallelism), message passing, distributed arrays, and single-assignment variables. Associated services ensure that resources are allocated to processes in a reasonable fashion, provide convenient abstractions for tertiary storage, and so forth.

There is no consensus on what programming model is appropriate for a grid environment, although it seems clear that many models will be used. Table 3 summarizes some of the models that have been proposed; new models will emerge as our understanding of grid programming evolves.

As Table 3 makes clear, one approach to grid programming is to adapt models that have already proved successful in sequential or parallel environments. For example, a grid-enabled distributed shared-
A wide range of new services can be expected to arise in grid environments to support the development of more complex grid applications. In addition to grid analogs of conventional services such as file systems, we will see new services for resource discovery, resource brokering, electronic payments, licensing, fault tolerance, specification of use conditions, configuration, adaptation, and distributed system management, to name just a few.
5.4 End users
Most grid users, like most users of computers or networks today, will not write programs. Instead, they will use grid-enabled applications that make use of grid resources and services. These applications may be chemistry packages or environmental models that use grid resources for computing or data; problem-solving packages that help set up parameter study experiments [1]; mathematical packages augmented with calls to network-enabled solvers [17], [11]; or collaborative engineering packages that allow geographically separated users to cooperate on the design of complex systems.

End users typically place stringent requirements on their tools, in terms of reliability, predictability, confidentiality, and usability. The construction of applications that can meet these requirements in complex grid environments represents a major research and engineering challenge.

5.5 System administrators
The final group of users that we consider are the system administrators who must manage the infrastructure on which computational grids operate. This task is complicated by the high degree of sharing that grids are designed to make possible. The user communities and resources associated with a particular grid will frequently span multiple administrative domains, and new services will arise—such as accounting and resource brokering—that require distributed management. Furthermore, individual resources may participate in several different grids, each with its own particular user community, access policies, and so on. For a grid to be effective, each participating resource must be administered so as to strike an appropriate balance between local policy requirements and the needs of the larger grid community. This problem has a significant political dimension, but new technical solutions are also required.

The Internet experience suggests that two keys to scalability when administering large distributed systems are to decentralize administration and to automate trans-site issues. For example, names and routes are administered locally, while essential trans-site services such as route discovery and name resolution are automated. Grids will require a new generation of tools for automatically monitoring and managing many tasks that are currently handled manually.

New administration issues that arise in grids include establishing, monitoring, and enforcing local policies in situations where the set of users may be large and dynamic; negotiating policy with other sites and users; accounting and payment mechanisms; and the establishment and management of markets and other resource-trading mechanisms. There are interesting parallels between these problems and management issues that arise in the electric power and banking industries [31], [28].

6. GRID ARCHITECTURE
What is involved in building a grid? To address this question, we adopt a system architect’s perspective and examine the organization of the software infrastructure required to support the grid users, applications, and services discussed in the preceding sections.

As noted above, computational grids will be created to serve different communities with widely varying characteristics and requirements. Hence, it seems unlikely that we will see a single grid architecture. However, we do believe that we can identify basic services that most grids will provide, with different grids adopting different approaches to the realization of these services.

One major driver for the techniques used to implement grid services is scale. Computational infrastructure, like other infrastructures, is fractal, or self-similar at different scales. We have networks between countries, organizations, clusters, and computers; between components of a computer; and even within a single component. However, at different scales, we often operate in different physical, economic, and political regimes. For example, the access control solutions used for a laptop computer’s system bus are probably not appropriate for a trans-Pacific cable.

In this section, we adopt scale as the major dimension for comparison. We consider four types of
systems, of increasing scale and complexity, asking two questions for each: What new concerns does this increase in scale introduce? And how do these new concerns influence how we provide basic services? These system types are as follows (see also Table 4):

Table 4: Computer systems operating at different scales

<table>
<thead>
<tr>
<th>Comp. model</th>
<th>I/O model</th>
<th>Resource manag.</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endsystem:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Multithreading, automatic parallelization,</td>
<td>Local I/O, disk-stripping</td>
<td>Process creation, OS signal delivery, OS scheduling</td>
<td>OS kernel, hardware</td>
</tr>
<tr>
<td><strong>Cluster</strong> (increased scale, reduced integration):</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Synchronous communication, distributed shared memory</td>
<td>Parallel I/O (e.g., MPI-IO), file systems</td>
<td>Parallel process creation, gang scheduling, OS-level signal propagation</td>
<td>Shared security databases</td>
</tr>
<tr>
<td><strong>Intranet</strong> (heterogeneity, separate administration, lack of global knowledge):</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Client/server, loosely synchronous: pipelines, coupling manager/worker</td>
<td>Distributed file systems (DFS, HPSS), databases</td>
<td>Resource discovery, signal distribution networks, high throughput</td>
<td>Network security (Kerberos)</td>
</tr>
<tr>
<td><strong>Internet</strong> (lack of centralized control, geographical distribution, int'l issues):</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Collaborative systems, remote control, data mining</td>
<td>Remote file access, digital libraries, data warehouses</td>
<td>Brokers, trading, mobile code negotiation</td>
<td>Trust delegation, public key, sandboxes</td>
</tr>
</tbody>
</table>

1. The *end system* provides the best model we have for what it means to compute, because it is here that most research and development efforts have focused in the past four decades.
2. The *cluster* introduces new issues of parallelism and distributed management, albeit of homogeneous systems.
3. The *intranet* introduces the additional issues of heterogeneity and geographical distribution.
4. The *internet* introduces issues associated with a lack of centralized control.

An important secondary driver for architectural solutions is the performance requirements of the grid. Stringent performance requirements amplify the effect of scale because they make it harder to hide heterogeneity. For example, if performance is not a big concern, it is straightforward to extend UNIX file I/O to support access to remote files, perhaps via a HyperText Transport Protocol (HTTP) gateway [66]. However, if performance is critical, remote access may require quite different mechanisms—such as parallel transfers over a striped network from a remote parallel file system to a local parallel computer—that are not easily expressed in terms of UNIX file I/O semantics. Hence, a high-performance wide area grid may need to adopt quite different solutions to data access problems. In the following, we assume that we are dealing with high-performance systems; systems with lower performance requirements are generally simpler.

### 6.1 Basic services

We start our discussion of architecture by reviewing the basic services provided on conventional computers. We do so because we believe that, in the absence of strong evidence to the contrary, services that have been developed and proven effective in several decades of conventional computing will also be desirable in computational grids. Grid environments also require additional services, but we claim that,
to a significant extent, grid development will be concerned with extending familiar capabilities to the more complex wide area environment.

Our purpose in this subsection is not to provide a detailed exposition of well-known ideas but rather to establish a vocabulary for subsequent discussion. We assume that we are discussing a generic modern computing system, and hence refrain from prefixing each statement with “in general,” “typically,” and the like. Individual systems will, of course, differ from the generic systems described here, sometimes in interesting and important ways.

The first step in a computation that involves shared resources is an authentication process, designed to establish the identity of the user. A subsequent authorization process establishes the right of the user to create entities called processes. A process comprises one or more threads of control, created for either concurrency or parallelism, and executing within a shared address space. A process can also communicate with other processes via a variety of abstractions, including shared memory (with semaphores or locks), pipes, and protocols such as TCP/IP.

A user (or process acting on behalf of a user) can control the activities in another process—for example, to suspend, resume, or terminate its execution. This control is achieved by means of asynchronously delivered signals.

A process acts on behalf of its creator to acquire resources, by executing instructions, occupying memory, reading and writing disks, sending and receiving messages, and so on. The ability of a process to acquire resources is limited by underlying authorization mechanisms, which implement a system’s resource allocation policy, taking into account the user’s identity, prior resource consumption, and/or other criteria. Scheduling mechanisms in the underlying system deal with competing demands for resources and may also (for example, in real-time systems) support user requests for performance guarantees.

Underlying accounting mechanisms keep track of resource allocations and consumption, and payment mechanisms may be provided to translate resource consumption into some common currency. The underlying system will also provide protection mechanisms to ensure that one user’s computation does not interfere with another’s.

Other services provide abstractions for secondary storage. Of these, virtual memory is implicit, extending the shared address space abstraction already noted; file systems and databases are more explicit representations of secondary storage.

6.2 End systems

Individual end systems—computers, storage systems, sensors, and other devices—are characterized by relatively small scale and a high degree of homogeneity and integration. There are typically just a few tens of components (processors, disks, etc.), these components are mostly of the same type, and the components and the software that controls them have been co-designed to simplify management and use and to maximize performance. (Specialized devices such as scientific instruments may be more significantly complex, with potentially thousands of internal components, of which hundreds may be visible externally.)

Such end systems represent the simplest, and most intensively studied, environment in which to provide the services listed above. The principal challenges facing developers of future systems of this type relate to changing computer architectures (in particular, parallel architectures) and the need to integrate end systems more fully into clusters, intranets, and internets.

State of the art

The software architectures used in conventional end systems are well known [61]. Basic services are provided by a privileged operating system, which has absolute control over the resources of the computer. This operating system handles authentication and mediates user process requests to acquire resources,
communicate with other processes, access files, and so on. The integrated nature of the hardware and operating system allows high-performance implementations of important functions such as virtual memory and I/O.

Programmers develop applications for these end systems by using a variety of high-level languages and tools. A high degree of integration between processor architecture, memory system, and compiler means that high performance can often be achieved with relatively little programmer effort.

Future directions
A significant deficiency of most end-system architectures is that they lack features necessary for integration into larger clusters, intranets, and internets. Much current research and development is concerned with evolving system end architectures in directions relevant to future computational grids. To list just three: Operating systems are evolving to support operation in clustered environments, in which services are distributed over multiple networked computers, rather than replicated on every processor [3], [65]. A second important trend is toward a greater integration of end systems (computers, disks, etc.) with networks, with the goal of reducing the overheads incurred at network interfaces and hence increasing communication rates [22], [35]. Finally, support for mobile code is starting to appear, in the form of authentication schemes, secure execution environments for downloaded code (“sandboxes”), and so on [32], [72], [71], [44].

The net effect of these various developments seems likely to be to reduce the currently sharp boundaries between end system, cluster, and intranet/internet, with the result that individual end systems will more fully embrace remote computation, as producers and/or consumers.

6.3 Clusters
The second class of systems that we consider is the cluster, or network of workstations: a collection of computers connected by a high-speed local area network and designed to be used as an integrated computing or data processing resource. A cluster, like an individual end system, is a homogeneous entity—its constituent systems differ primarily in configuration, not basic architecture—and is controlled by a single administrative entity who has complete control over each end system. The two principal complicating factors that the cluster introduces are as follows:
1. Increased physical scale: A cluster may comprise several hundred or thousand processors, with the result that alternative algorithms are needed for certain resource management and control functions.
2. Reduced integration: A desire to construct clusters from commodity parts means that clusters are often less integrated than end systems. One implication of this is reduced performance for certain functions (e.g., communication).

State of the art
The increased scale and reduced integration of the cluster environment make the implementation of certain services more difficult and also introduce a need for new services not required in a single end system. The result tends to be either significantly reduced performance (and hence range of applications) or software architectures that modify and/or extend end-system operating systems in significant ways.

We use the problem of high-performance parallel execution to illustrate the types of issues that can arise when we seek to provide familiar end-system services in a cluster environment. In a single (multiprocessor) end system, high-performance parallel execution is typically achieved either by using specialized communication libraries such as MPI or by creating multiple threads that communicate by reading and writing a shared address space.

Both message-passing and shared-memory programming models can be implemented in a cluster. Message passing is straightforward to implement, since the commodity systems from which clusters are
constructed typically support at least TCP/IP as a communication protocol. Shared memory requires additional effort: in an end system, hardware mechanisms ensure a uniform address space for all threads, but in a cluster, we are dealing with multiple address spaces. One approach to this problem is to implement a logical shared memory by providing software mechanisms for translating between local and global addresses, ensuring coherency between different versions of data, and so forth. A variety of such distributed shared-memory systems exist, varying according to the level at which sharing is permitted [76], [24], [53].

In low-performance environments, the cluster developer’s job is done at this point; message-passing and DSM systems can be run as user-level programs that use conventional communication protocols and mechanisms (e.g., TCP/IP) for interprocessor communication. However, if performance is important, considerable additional development effort may be required. Conventional network protocols are orders of magnitude slower than intra-end-system communication operations. Low-latency, high-bandwidth inter-end-system communication can require modifications to the protocols used for communication, the operating system’s treatment of network interfaces, or even the network interface hardware [70], [56].

The cluster developer who is concerned with parallel performance must also address the problem of coscheduling. There is little point in communicating extremely rapidly to a remote process that must be scheduled before it can respond. Coscheduling refers to techniques that seek to schedule simultaneously the processes constituting a computation on different processors [23], [63]. In certain highly integrated parallel computers, coscheduling is achieved by using a batch scheduler: processors are space shared, so that only one computation uses a processor at a time. Alternatively, the schedulers on the different systems can communicate, or the application itself can guide the local scheduling process to increase the likelihood that processes will be coscheduled [3], [14].

To summarize the points illustrated by this example: in clusters, the implementation of services taken for granted in end systems can require new approaches to the implementation of existing services (e.g., interprocess communication) and the development of new services (e.g., DSM and coscheduling). The complexity of the new approaches and services, as well as the number of modifications required to the commodity technologies from which clusters are constructed, tends to increase proportionally with performance requirements.

We can paint a similar picture in other areas, such as process creation, process control, and I/O. Experience shows that familiar services can be extended to the cluster environment without too much difficulty, especially if performance is not critical; the more sophisticated cluster systems provide transparent mechanisms for allocating resources, creating processes, controlling processes, accessing files, and so forth, that work regardless of a program’s location within the cluster. However, when performance is critical, new implementation techniques, low-level services, and high-level interfaces can be required [65], [25].

Future directions
Cluster architectures are evolving in response to three pressures:

1. Performance requirements motivate increased integration and hence operating system and hardware modifications (for example, to support fast communications).
2. Changed operational parameters introduce a need for new operating system and user-level services, such as coscheduling.
3. Economic pressures encourage a continued focus on commodity technologies, at the expense of decreased integration and hence performance and services.

It seems likely that, in the medium term, software architectures for clusters will converge with those for end systems, as end-system architectures address issues of network operation and scale.
6.4 Intranets

The third class of systems that we consider is the intranet, a grid comprising a potentially large number of resources that nevertheless belong to a single organization. Like a cluster, an intranet can assume centralized administrative control and hence a high degree of coordination among resources. The three principal complicating factors that an intranet introduces are as follows:

1. **Heterogeneity**: The end systems and networks used in an intranet are almost certainly of different types and capabilities. We cannot assume a single system image across all end systems.

2. **Separate administration**: Individual systems will be separately administered; this feature introduces additional heterogeneity and the need to negotiate potentially conflicting policies.

3. **Lack of global knowledge**: A consequence of the first two factors, and the increased number of end systems, is that it is not possible, in general, for any one person or computation to have accurate global knowledge of system structure or state.

**State of the art**

The software technologies employed in intranets focus primarily on the problems of physical and administrative heterogeneity. The result is typically a simpler, less tightly integrated set of services than in a typical cluster. Commonly, the services that are provided are concerned primarily with the sharing of data (e.g., distributed file systems, databases, Web services) or with providing access to specialized services, rather than with supporting the coordinated use of multiple resources. Access to nonlocal resources often requires the use of simple, high-level interfaces designed for “arm’s-length” operation in environments in which every operation may involve authentication, format conversions, error checking, and accounting. Nevertheless, centralized administrative control does mean that a certain degree of uniformity of mechanism and interface can be achieved; for example, all machines may be required to run a specific distributed file system or batch scheduler, or may be placed behind a firewall, hence simplifying security solutions.

Software architectures commonly used in intranets include the Distributed Computing Environment (DCE), DCOM, and CORBA. In these systems, programs typically do not allocate resources and create processes explicitly, but rather connect to established “services” that encapsulate hardware resources or provide defined computational services. Interactions occur via remote procedure call [33] or remote method invocation [55], [36], models designed for situations in which the parties involved have little knowledge of each other. Communications occur via standardized protocols (typically layered on TCP/IP) that are designed for portability rather than high performance. In larger intranets, particularly those used for mission-critical applications, reliable group communication protocols such as those implemented by ISIS [7] and Totem [46] can be used to deal with failure by ordering the occurrence of events within the system.

The limited centralized control provided by a parent organization can allow the deployment of distributed queuing systems such as Load Sharing Facility (LSF), Codine, or Condor, hence providing uniform access to compute resources. Such systems provide some support for remote management of computation, for example, by distributing a limited range of signals to processes through local servers and a logical signal distribution network. However, issues of security, payment mechanisms, and policy often prevent these solutions from scaling to large intranets.

In a similar fashion, uniform access to data resources can be provided by means of wide area file system technology (such as DFS), distributed database technology, or remote database access (such as SQL servers). High-performance, parallel access to data resources can be provided by more specialized systems such as the High Performance Storage System [73]. In these cases, the interfaces presented to the application would be the same as those provided in the cluster environment.

The greater heterogeneity, scale, and distribution of the intranet environment also introduce the need for services that are not needed in clusters. For example, resource discovery mechanisms may be
needed to support the discovery of the name, location, and other characteristics of resources currently available on the network. A reduced level of trust and greater exposure to external threats may motivate the use of more sophisticated security technologies. Here, we can once again exploit the limited centralized control that a parent organization can offer. Solutions such as Kerberos [51] can be mandated and integrated into the computational model, providing a unified authentication structure throughout the intranet.

**Future directions**

Existing intranet technologies do a reasonable job of projecting a subset of familiar programming models and services (procedure calls, file systems, etc.) into an environment of greater complexity and physical scale, but are inadequate for performance-driven applications. We expect future developments to overcome these difficulties by extending lighter-weight interaction models originally developed within clusters into the more complex intranet environment, and by developing specialized performance-oriented interfaces to various services.

### 6.5 Internets

The final class of systems that we consider is also the most challenging on which to perform network computing—internetworked systems that span multiple organizations. Like intranets, internets tend to be large and heterogeneous. The three principal additional complicating factors that an internet introduces are as follows:

1. **Lack of centralized control:** There is no central authority to enforce operational policies or to ensure resource quality, and so we see wide variation in both policy and quality.

2. **Geographical distribution:** Internets typically link resources that are geographically widely distributed. This distribution leads to network performance characteristics significantly different from those in local area or metropolitan area networks of clusters and intranets. Not only does latency scale linearly with distance, but bisection bandwidth arguments [18], [26] suggest that accessible bandwidth tends to decline linearly with distance, as a result of increased competition for long-haul links.

3. **International issues:** If a grid extends across international borders, export controls may constrain the technologies that can be used for security, and so on.

### State of the art

The internet environment’s scale and lack of central control have so far prevented the successful widespread deployment of grid services. Approaches that are effective in intranets often break down because of the increased scale and lack of centralized management. The set of assumptions that one user or resource can make about another is reduced yet further, a situation that can lead to a need for implementation techniques based on discovery and negotiation.

We use two examples to show how the internet environment can require new approaches. We first consider security. In an intranet, it can be reasonable to assume that every user has a preestablished trust relationship with every resource that he wishes to access. In the more open internet environment, this assumption becomes intractable because of the sheer number of potential process-to-resource relationships. This problem is accentuated by the dynamic and transient nature of computation, which makes any explicit representation of these relationships infeasible. Free-flowing interaction between computations and resources requires more dynamic approaches to authentication and access control. One potential solution is to introduce the notion of delegation of trust into security relationships; that is, we introduce mechanisms that allow an organization A to trust a user U because user U is trusted by a second organization B, with which A has a formal relationship. However, the development of such mechanisms remains a research problem.
As a second example, we consider the problem of coscheduling. In an intranet, it can be reasonable to assume that all resources run a single scheduler, whether a commercial system such as LSF or a research system such as Condor. Hence, it may be feasible to provide coscheduling facilities in support of applications that need to run on multiple resources at once. In an internet, we cannot rely on the existence of a common scheduling infrastructure. In this environment, coscheduling requires that a grid application (or scheduling service acting for an application) obtain knowledge of the scheduling policies that apply on different resources and influence the schedule either directly through an external scheduling API or indirectly via some other means [16].

**Future directions**

Future development of grid technologies for internet environments will involve the development of more sophisticated grid services and the gradual evolution of the services provided at end systems in support of those services. There is little consensus on the shape of the grid architectures that will emerge as a result of this process, but both commercial technologies and research projects point to interesting potential directions. Three of these directions—commodity technologies, Legion, and Globus—are explored in detail in later chapters. We note their key characteristics here but avoid discussion of their relative merits. There is as yet too little experience in their use for such discussion to be meaningful.

The commodity approach to grid architecture adopts as the basis for grid development the vast range of commodity technologies that are emerging at present, driven by the success of the Internet and Web and by the demands of electronic information delivery and commerce. These technologies are being used to construct three-tier architectures, in which middle-tier application servers mediate between sophisticated back-end services and potentially simple front ends. Grid applications are supported in this environment by means of specialized high-performance back-end and application servers.

The Legion approach to grid architecture seeks to use object-oriented design techniques to simplify the definition, deployment, application, and long-term evolution of grid components. Hence, the Legion architecture defines a complete object model that includes abstractions of compute resources called *host objects*, abstractions of storage systems called *data vault objects*, and a variety of other object classes. Users can use inheritance and other object-oriented techniques to specialize the behavior of these objects to their own particular needs, as well as develop new objects.

The Globus approach to grid architecture is based on two assumptions:

1. Grid architectures should provide basic services, but not prescribe particular programming models or higher-level architectures.
2. Grid applications require services beyond those provided by today’s commodity technologies.

Hence, the focus is on defining a “toolkit” of low-level services for security, communication, resource location, resource allocation, process management, and data access. These services are then used to implement higher-level services, tools, and programming models.

In addition, hybrids of these different architectural approaches are possible and will almost certainly be addressed; for example, a commodity three-tier system might use Globus services for its back end.

A wide range of other projects are exploring technologies of potential relevance to computational grids, for example, WebOS [67], Charlotte [6], UFO [2], ATLAS [5], Javelin [15], Popcorn [10], and Globe [69].

**7. RESEARCH CHALLENGES**

What problems must be solved to enable grid development? In preceding sections, we outlined what we expect grids to look like and how we expect them to be used. In doing so, we tried to be as concrete as possible, with the goal of providing at least a plausible view of the future. However, there are certainly
many challenges to be overcome before grids can be used as easily and flexibly as we have described. In this section, we summarize the nature of these challenges, most of which are discussed in much greater detail in the chapters that follow.

7.1 The Nature of applications
Early metacomputing experiments provide useful clues regarding the nature of the applications that will motivate and drive early grid development. However, history also tells us that dramatic changes in capabilities such as those discussed here are likely to lead to radically new ways of using computers—ways as yet unimagined. Research is required to explore the bounds of what is possible, both within those scientific and engineering domains in which metacomputing has traditionally been applied, and in other areas such as business, art, and entertainment.

7.2 Programming models and tools
As noted in Section 5., grid environments will require a rethinking of existing programming models and, most likely, new thinking about novel models more suitable for the specific characteristics of grid applications and environments. Within individual applications, new techniques are required for expressing advanced algorithms, for mapping those algorithms onto complex grid architectures, for translating user performance requirements into system resource requirements, and for adapting to changes in underlying system structure and state. Increased application and system complexity increases the importance of code reuse, and so techniques for the construction and composition of grid-enabled software components will be important. Another significant challenge is to provide tools that allow programmers to understand and explain program behavior and performance.

7.3 System architecture
The software systems that support grid applications must satisfy a variety of potentially conflicting requirements. A need for broad deployment implies that these systems must be simple and place minimal demands on local sites. At the same time, the need to achieve a wide variety of complex, performance-sensitive applications implies that these systems must provide a range of potentially sophisticated services. Other complicating factors include the need for scalability and evolution to future systems and services. It seems likely that new approaches to software architecture will be needed to meet these requirements—approaches that do not appear to be satisfied by existing Internet, distributed computing, or parallel computing technologies.

7.4 Algorithms and problem-solving methods
Grid environments differ substantially from conventional uniprocessor and parallel computing systems in their performance, cost, reliability, and security characteristics. These new characteristics will undoubtedly motivate the development of new classes of problem-solving methods and algorithms. Latency-tolerant and fault-tolerant solution strategies represent one important area in which research is required [5], [6], [10]. Highly concurrent and speculative execution techniques may be appropriate in environments where many more resources are available than at present.

7.5 Resource management
A defining feature of computational grids is that they involve sharing of networks, computers, and other resources. This sharing introduces challenging resource management problems that are beyond the state of the art in a variety of areas. Many of the applications described in later chapters need to meet stringent end-to-end performance requirements across multiple computational resources connected by heterogeneous, shared networks. To meet these requirements, we must provide improved methods for specify-
ing application-level requirements, for translating these requirements into computational resources and network-level quality-of-service parameters, and for arbitrating between conflicting demands.

7.6 Security
Sharing also introduces challenging security problems. Traditional network security research has focused primarily on two-party client-server interactions with relatively low performance requirements. Grid applications frequently involve many more entities, impose stringent performance requirements, and involve more complex activities such as collective operations and the downloading of code. In larger grids, issues that arise in electronic markets become important. Users may require assurance and licensing mechanisms that can provide guarantees (backed by financial obligations) that services behave as advertised [37].

7.7 Instrumentation and performance analysis
The complexity of grid environments and the performance complexity of many grid applications make techniques for collecting, analyzing, and explaining performance data of critical importance. Depending on the application and computing environment, poor performance as perceived by a user can be due to any one or a combination of many factors: an inappropriate algorithm, poor load balancing, inappropriate choice of communication protocol, contention for resources, or a faulty router. Significant advances in instrumentation, measurement, and analysis are required if we are to be able to relate subtle performance problems in the complex environments of future grids to appropriate application and system characteristics.

7.8 End systems
Grids also have implications for the end systems from which they are constructed. Today’s end systems are relatively small and are connected to networks by interfaces and with operating system mechanisms originally developed for reading and writing slow disks. Grids require that this model evolve in two dimensions. First, by increasing demand for high-performance networking, grid systems will motivate new approaches to operating system and network interface design in which networks are integrated with computers and operating systems at a more fundamental level than is the case today. Second, by developing new applications for networked computers, grids will accelerate local integration and hence increase the size and complexity of the end systems from which they are constructed.

7.9 Network protocols and infrastructure
Grid applications can be expected to have significant implications for future network protocols and hardware technologies. Mainstream developments in networking, particularly in the Internet community, have focused on best-effort service for large numbers of relatively low-bandwidth flows. Many of the future grid applications discussed in this book require both high bandwidths and stringent performance assurances. Meeting these requirements will require major advances in the technologies used to transport, switch, route, and manage network flows.

8. SUMMARY
This chapter has provided a high-level view of the expected purpose, shape, and architecture of future grid systems and, in the process, sketched a road map for more detailed technical discussion in subsequent chapters. The discussion was structured in terms of six questions.

Why do we need computational grids? We explained how grids can enhance human creativity by, for example, increasing the aggregate and peak computational performance available to important applications and allowing the coupling of geographically separated people and computers to support
collaborative engineering. We also discussed how such applications motivate our requirement for a software and hardware infrastructure able to provide dependable, consistent, and pervasive access to high-end computational capabilities.

**What types of applications will grids be used for?** We described five classes of grid applications: distributed supercomputing, in which many grid resources are used to solve very large problems; high throughput, in which grid resources are used to solve large numbers of small tasks; on demand, in which grids are used to meet peak needs for computational resources; data intensive, in which the focus is on coupling distributed data resources; and collaborative, in which grids are used to connect people.

**Who will use grids?** We examined the shape and concerns of four grid communities, each supporting a different type of grid: a national grid, serving a national government; a private grid, serving a health maintenance organization; a virtual grid, serving a scientific collaboratory; and a public grid, supporting a market for computational services.

**How will grids be used?** We analyzed the requirements of five classes of users for grid tools and services, distinguishing between the needs and concerns of end users, application developers, tool developers, grid developers, and system managers.

**What is involved in building a grid?** We discussed potential approaches to grid architecture, distinguishing between the differing concerns that arise and technologies that have been developed within individual end systems, clusters, intranets, and internets.

**What problems must be solved to enable grid development?** We provided a brief review of the research challenges that remain to be addressed before grids can be constructed and used on a large scale.

**FURTHER READING**

For more information on the topics covered in this chapter, see www.mkp.com/grids and also the following references:

- A series of books published by the Corporation for National Research Initiatives [29], [30], [31], [28] review and draw lessons from other large-scale infrastructures, such as the electric power grid, telecommunications network, and banking system.
- Catlett and Smarr’s original paper on metacomputing [13] provides an early vision of how high-performance distributed computing can change the way in which scientists and engineers use computing.
- Papers in a 1997 special issue of the *Communications of the ACM* [62] describe plans for a National Technology Grid.
- Several reports by the National Research Council touch upon issues relevant to grids [49], [50], [48].
- Birman and van Renesse [8] discuss the challenges that we face in achieving reliability in grid applications.

**References**


Agents - Mobile Agents in Java

M. Dönszelmann
CERN, Geneva, Switzerland

Abstract
The CERN School of Computing provided an excellent opportunity to try out mobile agents on physics analysis. The course explained the general concepts of mobile agents and their applicability to the field of High Energy Physics. In particular the students took a look at the merits of using mobile agents for physics analysis.

The course consisted mainly of lab works which provided both inexperienced and experienced students in the fields of Object-Oriented programming and Java an interesting introduction into the working of Mobile Agent Systems. A full Distributed Analysis System was put together.

1. Mobile Agents
Mobile agents[1] are programs that can move around on the network, while performing their duty, which may be a calculation, a database lookup or some other service. They keep their state as they move along from one machine to the next, thereby taking with them the result they have obtained so far. Machines not only provide a place for agents to work, but also a place for agents to meet and exchange information. A mobile agent may change its itinerary depending on the information it receives from another agent. Mobile software agents very much resemble the way people act, work and meet in the real world.

The fact that mobile agents can move themselves with their information across the network can reduce network traffic. For example, to discover information on another node, we would normally interrogate a server on that node via a mechanism such as remote procedure calls (RPC). We may have a set of questions, where later questions depend on the result of earlier ones. Using RPC every question (and its answer) would be separately transferred across the network. For mobile agents we use the term remote programming (RP). We instruct an agent with a set of questions and send it (with its questions) over to the remote node. The agent asks its questions locally and comes up with the final answer. It then travels back, only taking the final answer with it. Using RP, the network bandwidth consumed is the one of the agent and its final result, rather than all intermediate information of all the questions. We use the locality of information, to reduce the network bandwidth.

TeleScript[1], one of the oldest mobile agent systems, defines the concept of agent and place. An agent stays in a place and can do some work there. It may travel to another place, either by instructing itself, or being instructed by some other agent. Agents can meet other agents and communicate across the network to other agents. A place provides an environment for a agents to stay, to meet and to communicate. It provides security to protect the host system from hostile agents and to protect agents from each other. A set of collaborating agents, which can safely travel from one place to another and communicate with each other, may provide for a higher level service then the sum of the services provided by each of the agents individually.

Mobile agents have been around for some time, and many systems have been built to deploy them, such as TeleScript. The Java language[2] and its virtual machine seem to be very appropriate to implement an agent system. Java is platform independent, allows for serialization and persistency and comes with security built into the virtual machine. These are just some of the features needed to create an agent system. Several Java implementations exist today, one being the Aglet system[3] from IBM, which follows best the concepts of TeleScript.
2. COURSE SUMMARY

The course consisted of three parts: an introduction into agent system, implementations and examples, and an explanation of the exercises.

The introduction into agents mainly followed the book by W.T. Cockayne and M. Zyda[1]. The concept of mobile agents was explained using 21 students, who played a small game on stage. In this game 7 students were persons with some characteristics. 7 other students were tasks which had to calculate some number based on the characteristics of the persons. The final 7 volunteers were agents, which moved around on stage, gathering information from the persons, doing the calculation and bringing the result back to their task.

The implementations of agent systems using the Java platform was discussed in depth, including some smaller examples such as a distributed web-search, which uses mobile agents for its search strategy and its results. In the area of High Energy Physics three examples were given: a smart e-mail system, in which e-mail is in fact a mobile agent gathering updates on a physics paper, a slow controls system in which alarms of different severity are sent around as mobile agents, who then meet and decide if a real alarm should be triggered, and a distributed physics analysis system, which the students were supposed to built during the lab works.

The distributed physics analysis system uses mobile agents for physics jobs. This enables us to make the job travel from one data set to another, without moving the actual data. The data sets are assumed to be in different places. The job travels with its set of histograms (results), and fills them with information from each data set. Note that this is different from a job submission system, in which jobs cannot pack up and move to the next data set, and results have to be merged afterwards. An optimization could take place having agent jobs run in parallel on different data sets. Results would still have to be merged, but that responsibility is now with the agent job.

3. LAB WORK SUMMARY

The exercises consisted of three parts. You could choose between Part-A or Part-B, followed at all times by Part-C. The documents which describe the exercises are available from the agents web site [4].

3.1 Part - A

Part-A was meant for the non-java and non-object-oriented programmers. No programming knowledge was required. Some explanation on Object Oriented Programming was provided. The description in the document for part A was fairly explicit. A step by step description on what to do, using examples, would tell the student how to reach his end goal.

As a first exercise one created a set of three simple classes, and made a small calculation. As a second exercise an Analysis Job was programmed, thereby making use of pre-fabricated classes from an Analysis library. It was this Job which was to be handed to the group doing Part-B, who would run it in their Distributed Analysis System. Optionally the exercise was extended to create some agents which travelled around and did some calculation.

3.2 Part - B

Part-B was meant for the java or c++ experienced. The goal of part B was to write a distributed analysis system, which would take a job of part A and move it around on the network. An Agent System (Aglets) was provided, but some Agent classes had to be written. The description in the document for part B was in the form of requirements for classes and hints on how to implement these. There was no step by step description provided, assuming the students would know how to go about.

3.3 Part - C

At the end of the course, the whole system would be exercised on the machines in the lab. One would actually try to measure network traffic.
4. CONCLUSIONS
A lot of things went right during the exercises, the most important part of the course. Most people started doing part-A, some with little knowledge on object-oriented programming, learning Java, understanding Agents and most groups finished it, including the optional extension.

Part-B was done by a smaller group, including some who finished early with part-A. Here is where some things went wrong. Two problems in the agent system were discovered, one to deal with naming, the other with security. It was thought that the Aglet system was still pretty fragile to use for production.

Only a few groups managed to get as far as Part-C, mainly due to the bugs discovered. They run their system over several machines analyzing data by moving agents across the network, rather than moving data. No network measurement was performed though.

In a feedback session, held at the end of the course, it was thought it would be a nice idea to try the same exercise, running agents world-wide, on the machines of the students at their work-place. A decision was taken to set up a web site for this, to wait for an upgrade of the Aglet system, so that the bugs found earlier would be fixed and to try this out before the next school. Information can be found on the agents web site [4].

5. ACKNOWLEDGMENTS
I would like to thank those people who helped making the course on Agent System a success. In particular I thank Paolo Palazzi, my former boss, for giving me the idea to teach about agent systems at the school and Gottfried Kellner for letting me spend time on the preparation of it. A team of people, Arash Khodabandeh, Alberto Aimar, Klaus Ackerstaff and Eberhard Wolff were present during the exercises and lectures and helped out with the logistics and organization. Mario Ruggier and Bertrand Rousseau were both helping behind the set in providing document templates and logistics at CERN. Andreu Pacheco and Fabien Collin provided a stable set of machines on which we could do our exercises. Thanks to all of them. However, I should not forget to thank all the students of the school for their active participation, including the summer students of 1998 at CERN, who tried out the course in a mini version. You all made this course a great success.

6. REFERENCES
The production of software is a labor intensive activity, especially in the field of High Energy Physics, where the complexity of current and future experiments require large software projects.

For most of the scientists and engineers involved in software production, the business is science or engineering, not computing. As software scope continues to grow so does the feeling that its development and maintenance are out of control. The situation is made even worse by a lack of software engineers and from an uneven software culture. Better organization and control of software production are clearly needed in order to face the challenge of developing new software for the LHC and of maintaining software for previous experiments.

To be able to control the production of software it is essential to improve (a) the knowledge of the PEOPLE involved, (b) the organization and improvement of the software development PROCESS and (c) the TECHNOLOGY used in the various aspects of this activity. The goals are better systems at lower cost, and of better quality.

The process is the set of orderly actions to be performed to produce the software throughout the life cycle. The quality of a process can be measured in terms of maturity against a recognized framework. A reference framework is the Capability Maturity Model (CMM) proposed by the Software Engineering Institute (SEI). This model consists of five levels and the most difficult step is to move from level 1 to level 2 because of all the management procedures and activities that have to be put in place. As an organization moves up the maturity levels, management visibility on the software process improves, estimates become more accurate, schedules are met more precisely and the time required to produce a software system shortens. This progression in maturity levels is also called Software Process Improvement (SPI).
Except for level 1 (where all organizations start), each maturity level is decomposed into several Key Process Areas (KPA) that indicate the actions an organization should focus on to improve its software process and reach the corresponding level. The Software Evolution track uses the level 2 as a file rouge. Every lecture teaches one level 2 KPAs:

- User Requirements by G.Kellner (CERN IPT Group)
- Project Management and Tracking by A.Aimar (CERN IPT Group)
- Configuration Management by B.Jacobsen (UC Berkeley) and D.Johnson (Univ. Colorado)
- Quality through Software Metrics by A.Khodabandeh (CERN IPT Group)

The track also contains one lecture on documentation, an important aspect of software engineering too often overlooked:

- Software Documentation by A.Aimar (CERN IPT Group)

Two case studies illustrate the adequacy of the presented concepts to our environment by showing how they have been put into practice in HEP experiments:

- BaBar Case Study by B.Jacobsen (UC Berkeley)
- OPAL Online - A Case Study by K.Ackerstaff (CERN OPAL)

The track consists of 8 hours of lectures, 11 hours of exercises and wrap-up, and 4 hours of case studies.

REFERENCES

USER REQUIREMENTS

Gottfried Kellner  
CERN, Geneva, Switzerland

Abstract
The requirements are a collection of statements that should describe in a clear, concise, consistent and unambiguous manner all significant aspects of a proposed system. They should describe what the computer system should be capable to do and describe the environmental need which the proposed software is to satisfy. User requirements definition is an iterative process to find the widest possible agreement through interviews and surveys. The overriding objective of requirements analysis is to provide necessary and sufficient information for subsequent design, implementation, and validation & verification to be successful. Requirements form the baseline for system development, when the system is initially built, but also through subsequent maintenance and enhancement. We will discuss the problems involved in requirements gathering and analysis, the content and property of a good requirements specification, and a number of general and specific approaches to requirements analysis.

1. INTRODUCTION
Already since the mid-70s it is known that requirements errors are the most numerous and, more significantly, that they also are the most costly and time-consuming to correct. The recognition of the critical nature of requirements established Requirements Engineering as an important sub-field of Software Engineering. A number of research activities and conferences are devoted to this field since many years [1].

Quoting from Fred Brooks [2]:

- The hardest single part of building a software system is deciding precisely what to build.
- No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all the interfaces to people, to machines, and to other software systems.
- No part of the work so cripples the resulting system if done wrong.
- No other part is more difficult to rectify later.

Results of industry studies in the 1970s described by Boehm [3], and since replicated many times, showed that requirements errors are the most costly. These studies all produced the same basic result: the earlier in the development process an error occurs and the later the error is detected, the more expensive it is to correct. Moreover, the relative cost rises quickly – an error that costs one dollar to fix in the requirements phase may cost $ 20 to fix during unit tests, and $ 100-200 to fix if not corrected until the system is put into operation or in the maintenance phase.

2. IMPORTANCE OF (USER) REQUIREMENTS
Deciding precisely what to build and documenting the results is the goal of the requirements phase of software development. The traditional “waterfall model” [4] of the software life-cycle assumes that at the end of the user requirements phase a requirements specification exists which is unambiguous,
complete, consistent, verifiable and validatable. This specification should establish and specify precisely what the software must do without describing how to do it. This rather simple-minded approach might be applicable for rather small and well-defined software projects. In most cases requirements are not fully known, or are not clearly understood. An iterative approach, including prototyping or partial design may be needed to clarify issues. A variety of software life-cycle models have been evolved which better address the inherent difficulty to arrive at requirements specifications – incremental, prototyping, RAD, spiral, 4GL, and others [4]. Even if the specification document would be perfect to start with there will always be changes due to planned upgrades or unanticipated changes to functionality or environment of the system.

Good requirements are essential if we are to be sure that we are building the system the user wants, and that we are not doing more than is needed. They should play this role not only during initial development, but also during subsequent maintenance and enhancements. In practice there are rather few systems where the requirements are good enough to be useful throughout the system’s life. Requirement management is one of the key process areas to achieve this goal.

3. USER REQUIREMENTS

User Requirements are the primary communication vehicle between users (and customers) of a desired software system and the computer specialists to document the user’s perceived needs. They describe both the system and the environment in which it will operate. They must provide necessary and sufficient information for subsequent design and implementation to be useful.

A requirement is a ‘condition or capability needed by a user to solve a problem or achieve an objective’ [5]. This definition leads to two principal categories of requirements: ‘capability requirements’ and ‘constraint requirements’ [6]. Capability requirements describe the process to be supported by software. They should define operations, or a sequence of related operations, that the software will be able to perform. Capability requirements should be qualified with attributes, where feasible (e.g. capacity, speed, accuracy). Constraint requirements place restrictions on how the user requirements are to be met. The user may place constraints on the software related to interface requirements (e.g. communication, hardware, software, human-computer interaction), to quality requirements (e.g. adaptability, availability, portability, security, safety, standards), or requirements on resources and time-scales for producing and operating the software.

4. USER REQUIREMENTS DEFINITION

A variety of methods and tools are used to elicit (i.e. capture), specify and document the user requirements. Considering the importance of user requirements rather ‘low-tech’ techniques are used. Requirements elicitation is still a very active field of research in computer science, cognitive psychology, social science, Artificial Intelligence, amongst others. Many papers are presented at conferences, but few of these ideas are used in real-life projects outside the study cases.

People involved in the elicitation process are users, customers, designers and software engineers, project managers, and facilitators with extensive knowledge in techniques for elicitation, analysis and design of software. The purpose is to gather a maximum of information about user needs, to capture the user’s aims and objectives, and to clarify requests or statements which are ambiguous, incomplete, too detailed, etc. Methods used include interviews, surveys, studies of existing systems, feasibility studies, exploratory prototyping, joint or rapid application development procedures, identification of scenarios, use cases or patterns. Usually a combination of these techniques is applied, depending on the project, the expertise of the users to identify and express their needs, and the skills of the facilitator.
Most user requirements are specified in simple natural language. These can be complemented by diagrams, tables, mathematical formalisms, context diagrams for describing the environment, indeed anything that can help in clarification of issues. Word processors, spread-sheet and diagramming tools are mostly used. For larger software systems database management systems and requirements engineering tools will help to manage the complexity of a large number of requirements and their interdependency. These tools are particularly useful to cope with problems of requirements management over long periods of time for development and operation of software systems.

Documentation of user requirements, the User Requirements Document (URD), usually applies standard templates provided by a variety of sources. We have adopted the layout proposed by the ESA PSS-05 documents. ESA PSS-05 provides standards and guidelines for the whole software life-cycle and is widely used in industry in Europe [6]. A standard URD template using Adobe FrameMaker [7], as well as several examples of URD for HEP applications have been made available on the web [8]. A URD will normally pass many cycles of internal reviews for updates and clarifications. Acceptance tests will be identified for validation of the user requirements in the transfer phase at the end of the software development cycle. A formal review of the URD and the Acceptance Test Plan, involving users, developers, management and QA staff, will conclude by a formal statement that the project is ready to proceed. Formally, the software life-cycle begins with the acceptance of the User Requirements Document.

5. SUMMARY

Requirements are intrinsically hard to do well. Beyond the need for discipline, there are a number of difficulties that attend both the understanding of requirements and their specification. Technical and human concerns have to be addressed to manage complexity or communicate to different audiences. All of the approaches outlined very briefly above have significant weaknesses, but experience confirms that the use of any careful and systematic approach is preferable to an ad-hoc and chaotic one. Non-existent or inconsistent requirements will end up in poor quality software and expensive rework. Benefits of good requirements come at a cost. It needs people with adequate experience, training and resources at the begin of a project, not after disaster has struck. The bulk of software problems arise from inadequate specifications, not from errors in implementation.

REFERENCES


The Mythical Man-Month, F.Brooks, Addison-Wesley, 1975


[8] see http://www.cern.ch/FrameMaker/#templates
PROJECT MANAGEMENT AND TRACKING, RESOURCE ESTIMATE

Alberto Aimar
IPT Group, IT Division
CERN, Geneva, Switzerland

Abstract
Participants of software projects often live in a permanent state of agitation and worry (or their supervisors do). This is mainly due to the ignorance of the factors determining success, or failure, of a project.

In order to reduce these uncertainties, and be able to improve plans and estimations, a minimal effort in getting the project and its people organized is crucial.

General project management and monitoring techniques can be adapted to projects of any size and time-scale. This document highlights the key elements in planning and tracking small projects, and suggests practices that can be of immediate benefit to the reader.

1. INTRODUCTION
Project management is an activity that is performed during the whole of the software lifecycle. There are three fundamental aspects involved in the management of a project:

- the organization of the project and of the resources;
- the human aspects among the members of the project and the relationship with the project environment;
- the project application field and the technology used during the project.

The presentation and this summary will give an overview of the first two elements, the latter one is related to the specific project and depends on the application developed and its architecture (modularity, distribution, application logic, etc.).

The suggestions contained in this paper are intended for projects of all sizes, even one-member projects and you are encouraged to try these suggestions, little by little in your daily work and in your current projects. There is nothing magic (or evil) in project management, just the attempt to organize our work better in order to get to the project goal with a smaller effort. Do not wait for a big project to manage: before driving a bus filled with fifty people, your are expected to be able at least to drive your own car!

2. WHAT IS A PROJECT?
All companies or even individuals and, more in general, all organizations perform on-going activities that are there to create products or to provide services that are the reason for their business. While these organizations keep performing the same regular operations, producing the same products and providing the same services, they do not engage in any project. A “project” is synonymous of “change” in these regular trends in an organization.

Projects are launched to start new activities, produce new artifacts or improve the current products and services. Projects are temporary organizations created to achieve a precise common goal. They typically involve people(who), resources(what) and tasks(why) to perform to achieve the defined goal (why).

Examples of projects are: building a HEP experiment, preparing a talk for the CSC, a holiday trip, creating a new software application, etc. On the contrary, activities like running an HEP experiment or research activities are not projects. Research activities are not projects in themselves, they can be defined in term of projects but the research part, as such, is not a project.
2.1 Project and product lifetime

A project is a temporary organization even if it creates a new product/service that will continue well beyond the life of the project. The separation between the phases in Fig. 1 must not be taken as being “carved in stone”. In fact, changes to the definition of the product can always be done, even after the start on the project, but they must be controlled with predefined procedures and with a clear understanding of their cost.

But these phases must exist in order not to proceed to the development of the product in a confused and uncontrolled way.

Before a project is started the product must be studied in detail and while the details become clearer a “project plan” can be conceived (Fig. 1).

![Fig. 1 Difference between project and product lifetime.](image)

The product study is the crucial part of the definition of the product: if it is too imprecise the project will be under constant rescheduling and change, if it is too long the project will start too late and no changes will be possible. Once the plan is detailed enough and the “project life” starts, then the leading and scheduling activities take place for the whole life of the project. When the project ends, the “product life” starts and activities such as support and maintenance start. They are not part of the project, their definition can be one of the project tasks (or of other projects) but their execution is not in the project.

2.2 Project and product management roles

The product and the project have also very different management aspects. Figure 2 shows the relationships among the individuals (or organization bodies) from the launch until the execution of a project.

![Fig. 2 Project management roles.](image)

The “product owner” is the one that wants the product and s/he decides if the product is worth building by studying three factors such as: the costs involved, the value of the product and the risk that
the costs will be greater than the value. When the product is considered worth building the owner will ask a “product manager” to define it in detail (like an architect for the building of a house). Together they will contact a “project owner” that will take charge of building the product.

The project owner will decide if s/he can build the product in term of the following factors: budget allocated for the project, schedule for the production and quality required by the client. If the development of the project is convenient for the owner s/he will appoint a “project manager” that will define and lead the project. The project manager will discuss a project plan with the product manager for the aspects concerning the product (quality, features, etc.) and with the project owner for the resources (people, equipment, budget, etc.).

3. PROJECT SCHEDULING

The project scheduling activities have one unique purpose: the definition of the “project alignment” of the tasks to perform. In order to do this the project manager receives or has defined previously:

- the tasks to perform and their dependencies (order, priorities, etc.);
- the resources available and their constraints (other commitments, holidays, etc.).

As result of the scheduling activity the project manager must produce:

- the project timetable
- the assignments of the resources.

A graphic representation of a project plan is usually a Gantt chart (see [1]) that distributes the tasks over time and assigns the resources to the tasks. The constraints of the resources are very important for the correct scheduling of the project tasks. In Fig. 3, for instance, the dependency between tasks would allow the execution of a task (“taskA”) at an earlier time but the need to have the resources available (“rsr2” and “rsr3”) forces the task execution to a later moment.

![Fig. 3 Project alignment without and with the resources alignment.](image)

After the definition of the plan (Fig. 4) if the delivery date is:

- in the “go area” then the delivery date is after the date resulting from the scheduling; in this case the resources allocated are sufficient;
- in the “warning area” it means that with the resources constraints it is not possible to respect the delivery date; but with more resources the goal is achievable;

![Fig. 4 Project target date ignoring and considering resources constraints.](image)
in the “stop area”, where only the dependency of the tasks are considered, then there is no chance of achieving the success of the project: the goal must be reduced.

4. PROJECT TRACKING

The project tracking is the activity of the project manager in which s/he follows the project in order to take actions, if they are needed. Tracking means to:

- collect the data of the project;
- see what has been performed and has still to be performed;
- produce a progress status report periodically;
- estimate the time to complete and take the necessary actions if this time is beyond the scheduled delivery date.

4.1 Project progress

The progress status must be calculated for each task. There are three types of progress that can be calculated: technical progress (produced/total units), effort progress (actual/estimated effort), duration progress (actual/estimated duration). The technical progress is the one that allows the tracking of the project, for this reason it is necessary to find the right units to measure each task (product units, parts of product, dollars spent, etc.) and to define as many milestones as possible. Milestones are mandatory for real tracking. The effort and the duration progress allow the project manager to judge: how did s/he correctly estimate the effort needed, in terms of resources or of time needed.

The global completion of the project is then derived from the weighted combination of the progress (pr1, pr2, pr3, etc.) of the individual tasks. Usually the weight (w1, w2, w3, etc.) is based on importance, complexity or budget of each task (total progress = pr1 w1 + pr2 w2 + pr3 w3 + etc.).

4.2 The Milestone Trend Chart

The Milestone Trend Chart (MTC) in Fig. 5 displays the periodical recalculations of the milestones compared with the initial estimates and shows the trend of the project (the milestones are met when they “cross” the diagonal line on the chart) and the accuracy of the estimates (if the milestones do not move vertically on the chart).

5. PROJECT PLANNING

Scheduling and tracking are activities that can be performed only after a project plan has been specified. A plan is not a wishful prediction, it is a plan accepted by all parties involved in the management of the project and of the product. With a plan one will, especially at the beginning, fail by 30-40%; without a plan one will not get the product at all, one will not get the product desired and not within the budget one was supposed to spend.

5.1 Plan definition

Before the definition of the plan only the following are known: 1-the initial status: the intention to proceed in the building of the product; 2- the final status: the product that does not yet exist, but has been identified during the requirements phase and the product study.

How do you plan how to go from the initial to the final status?

- Do brainstorming, involve the members of the projects, really consider all aspects and possible problems
- Identify all the details. I really mean it: “think of ALL the details you can imagine!” and find all deliverables, actions, tasks, internal milestones, training needed, skills required, etc.
- Define the milestones and the actions, then link the milestones with the actions that lead to them.
- Remove all actions that do not lead to milestones (it can happen, during the brainstorming solutions are proposed and then discarded).
- Build the network of actions (usually in several graphs like the one in Fig. 6).
Fig. 5 The Milestone Trend Chart.

Fig. 6 The Milestone-Action network.
6. STRUCTURED PROJECT MANAGEMENT

The structured project management method is proposed by Fergus O’Connell [1] and drives the project manager, in human motivation and in project organization, by suggesting the pragmatic and consistent execution of ten steps. I will briefly go through all the steps described in the book, but this reference is really worth buying and using daily if you want to be a successful project manager.

**Step 1. Visualize the goal**

This first step is crucial and not obvious: one has to identify the goal as precisely as possible. All techniques mentioned previously (brainstorming, daydreaming, etc.) are valid; make a checklist of all the deliverables. Share your vision with your team, build it with them, make it a common goal.

Of course there will be changes! What is important is that these changes are controlled, studied, and their impact must be considered in the project plan. A project plan is always function of four factors: functionality, delivery date, effort and quality. Any change in one of these factors influences the others: for instance one cannot add a feature and keep the other three unchanged, they must be re-evaluated.

**Step 2. Make a list of the jobs to be done**

Make a check list of the tasks to perform and for each of them define:

- resources required (equipment, products, services) and skills required (hiring or training);
- identify the milestones and the tasks, the dependencies among tasks and high-risk tasks;
- write down who is responsible for the task.

Two comments: 1- In order to do and maintain this list, use a planning tool, even a bad one is better than none. 2- Is this a pain in the neck? You bet, yes it is! But it is in this step that you “decide” if your project will succeed or fail!

**Step 3. There must be one leader**

The leader is not the one that takes the reward for all that happens in the project and talks to the bosses. S/he is the one that is responsible for getting the project done; nothing more (or less) than that. S/he is the one that measures the pulse of the project and that also give the impetus to the project; the leader must be a role model for the other project members and must be able to accept full responsibility and be ready to share the success of the project.

One leader will not ensure success. More that one leader will ensure trouble, fights and sure failure. For this reason make sure that all roles are clear and do not hesitate to delegate part of the project if you trust the people and if you can make sure that they are adequate leaders of sub-projects.

**Step 4. Assign people to jobs**

Each job now has a name and it is defined (step 2). The people in the team need to find their own professional goals in the projects; for this reason you must estimate with them the complexity and the tasks. Make sure they are as committed as you are or that at least they understand that they will benefit from participating in the project. Do not forget people’s expectations about training and personal development (10%), their other commitments and their project management time (10%).

**Step 5. Manage expectations and allow margins for errors**

One fact you know for sure: your plan is definitely not right! Therefore add a special task called “contingency” (about 20%); actually mature organizations require this task: it would be crazy to think that we can foresee the future with higher precision. If you cannot add an explicit contingency, then hide it with specific and slightly cryptic tasks.

Contingency can be in any of the four variables (functionality, delivery date, effort and quality) and when you discuss the project really show all the details of your plan and be ready to discuss all the variables, but do it calmly and with the people that want the product. If you feel that you cannot do the project with the means that are given to you, the best you can do is to state it, explain your reasons and see if you can reduce the features expected in the product.
Step 6. Appropriate leadership style
Use an appropriate leadership style. This means that you must be the model of behavior for the members, reduce meetings to the strict necessary and deal with individual problems in private. Follow people closely only if necessary but give more and more freedom when you trust them, acknowledge their goals and that you care about them as much as you expect them to care about the project.

Step 7. Know what it going on
Be always informed of what happens inside and around your project: you must know it before it spreads in the project team. Good signs are fun, common vision, few crises, and external positive feedback. Bad signs are low morale and personal clashes, external rumours, missed milestones or forgotten tasks showing up.

Step 8. Tell people what is going on
Prepare regularly a status report and at least answer these questions: Is the project on schedule? Has anything new been added? What actions and changes have been performed?

Inform the people inside the project via a detailed status report, They must know exactly what is going on, why changes take place and what problems arise. In this way they will feel you trust them and you will often receive useful suggestions and solutions from them.

Inform the people outside the project. Provide them with a clear summary and do not hide changes or problems, they will find out anyway from rumours or missed deliveries. If you inform them you will be judged as being a responsible project manager, if you hide things now you are looking for bigger problems later, when it will be too late.

Step 9. Repeat steps 1 to 8
How often do you perform the previous steps? Every month? Every week? Every day? Yes, every month, every week and every day!

You are always checking milestones, looking ahead for problems, refining the job lists, following the morale of the team and so on. Software engineering is still a very young engineering field and cannot be compared, for instance, with civil engineering where there are formulas to calculate how long it takes to build a wall of a given size or to put a pillar in place. For this reason we need to follow our project always and very closely.

Step 10. The prize: the goal is achieved!
Here the project ends. But not your work. You have to recollect all your memory and data in a document. You must trace what went right and what went wrong. What surprises occurred? How much contingency did you use? Were people expectations achieved? Were your customers satisfied? What did you learn? And one question that is the most important for me: “would these people still work with me in my next project?”.

7. CONCLUSIONS
This was a quick overview of the technical and human aspects involved in project management. Try the suggestions above in your current project (even if you are the only project resource!). Define carefully your project goal and plan, schedule your work, track your progress and keep track of the project status. Make the effort of trying some of the ten steps, you will greatly benefit from them.

8. RECOMMENDED READING
INTRODUCTION TO SOFTWARE DOCUMENTATION

Alberto Aimar
IPT Group, IT Division
CERN, Geneva, Switzerland

Abstract
Documentation and information systems are needed in all phases of the development of a software product: from the user requirements phase, through design and implementation, until delivery and maintenance of the product. For this reason documentation should be designed, managed, produced, published and maintained. These activities are often seen as second priority or a burden, but are extremely useful. They become easier when performed in a coherent environment that allows production of adequate artifacts and their publication in paper and electronic forms. This summary provides some practical guidelines and references to resources that can help in the production of simple but effective documentation.

1. INTRODUCTION
During the development of a software project, documentation is undoubtedly the activity that we all like to do the least. On the other hand, often programs are simply thrown away and rewritten because they are no longer under control and there is not enough information to understand them. This summary is an attempt to provide recipes that reduce the pain of producing that minimal amount of documentation that will help other people to work on the same project (internal documentation) and that will allow somebody outside the project to use the product (user documentation).

2. DOCUMENTATION AND THE SOFTWARE LIFECYCLE
The process of producing software is made up of several activities (project management, configuration management, etc.) and phases (requirements, design, coding, etc.). Such activities and phases are always present in the lifecycle, independently on the development model chosen for the project. Figure 1 shows the “waterfall” and the “V” models, but there are also other models such as the “iterative” and the “rapid prototyping” models.

Fig. 1 The waterfall and the V software development models.
The purpose of this summary is not to discuss which model is the best: different models suit better different organizations or projects; what all these models have in common, and I intend to talk about this here, is that any software product without solid documentation will not last long or will cost much more than expected to maintain. Without information on the development process, implementation choices, test criteria, etc., nothing is clear and the software project always ends up in an unmanageable mess. What I call “solid documentation” is not the production of tons of paper but the updated and structured amount of information needed to keep the project under control.

Some projects will focus more on some phases and skip others; but documents must be produced during the development of a project. As shown in Fig. 2, all phases are done by different people (sometimes the same people, if they cover more than one role in the project) using different tools to produce different kinds of documents. In fact, the arrows in Fig. 1 are the documents that are flowing in the software development process.

![Fig. 2 Some of the documents produced in the software development lifecycle.](image)

3. WHY DO WE NEED BETTER DOCUMENTATION?

There is clear fact: writing documentation is not fun: it is a need. As Fig. 3 shows, in a software project 60% of the time is spent to understand and redo (27% + 33%) existing programs. This is often due to the fact that the documents supporting the project are inadequate, not detailed enough or obsolete.

![Fig. 3 The Grady experience: cost distribution of software development.](image)

Poor documentation increases the time not spent in producing new features, poorly documented products have difficulty in finding users and devolve slowly until they become “untouchable” and need to be partially rewritten or simply thrown away. Good documentation, on the other hand, helps people
to quickly integrate in the project, allow proper project management because problems are highlighted early and the development can be controlled adequately.

Inside a project the documentation is the glue between phases and among the people working together, at different artifacts, but with the same project goal. The quality of a product is something that must be built, it is not something that comes for free and without effort.

The documentation of the early phases of the project (requirements and analysis) is often neglected but actually it is in these early phases where correcting wrong choices is much cheaper. As Fig. 4 shows, a wrong choice or a forgotten detail found in the implementation phase costs up to hundred times more than if it had been found in the analysis phase.

![Fig. 4 Cost of a change in the different phases of the software lifecycle.](image)

4. SOFTWARE ENGINEERING DOCUMENTATION

Software engineering is still a very young field compared to others, such as civil engineering or mechanical engineering. Would it be thinkable to build a bridge or a new car model without properly describing its components? And to proceed immediately to build bits and pieces thinking to put them together afterwards, without a real design and assembly plans? No! So, why do we do it in software?

There are several standards defined by the main software engineering institutes (for more information see Section 8.3) but the purpose here is to highlight very briefly the characteristics of the different documents (the grey spots Fig. 5) and not to present all the details.

![Fig. 5 Basic documents in the software development lifecycle.](image)
4.1 Requirements documents

The requirements documents must describe the product, the functionalities and the quality expected. Wrong requirements will give the wrong product, fuzzy requirements will cause constant instability and changes.

In order to capture all aspects of the product, the requirements are usually separated in:
- functional requirements (features, deliverables, logic, documents, etc.);
- design requirements (quality, portability, performance, etc.);
- operational requirements (maintenance, security, support, etc.).

Each user requirement is formed by:
- a unique identifier;
- a description that must be unambiguous and clearly stated;
- attributes that complete the description, such as priority, examples, source, how to test the requirement, its acceptance criteria, etc.

See also the “User Requirements” paper in these proceedings.

4.2 Architecture documents

The architecture of a software artifact is its description in terms of all its components and the interfaces among the components. Usually the overview is done graphically (example in Fig. 6) but then the details must be specified in detailed text, in an architecture document.

Each component must have a unique identifier, the description of all its functionalities and its decomposition, if necessary, in sub-components. The interfaces are extremely important for obtaining modular applications, an efficient partition of the work in the team and parallel development. Each interface must be specified exactly in terms of data exchanged, direction of the data, which component triggers the data exchange, the detailed description of the data structures involved.

Fig. 6 Graphic overview of a component and its context.

4.3 Detailed Design

Once the components are defined, it is necessary to describe all the details of the component; usually for this purpose an analysis and design methodology (for instance OMT, Booch, UML, ROOM for object-oriented modeling) is used and its choice depends on the organization, the project infrastructure and the language platform chosen. Depending on the type of component these documents usually consist of use cases or scenarios, object or data models, dynamic or logic behavior models and so on.

4.4 Implementation documents

These are the result of the favorite activities of every developer: coding, debugging, etc. But code is not all, it must be the implementation of the design choices, must be adequately commented (I know these things are obvious but often they are not done!). For instance, comments should focus on the difficult
parts of the code, not simply parrot the easy bits of code; they should also explain the logic and why such logic is chosen.

4.5 Test documents

The product must be tested in order to make sure that it is “the” product we wanted to obtain; usually the test documents are done in parallel with coding and they use as input the requirement and the design documents. Tests, just like the application, must be studied, designed and implemented.

Usually the minimal set of tests are:

• a test plan that describes the global strategy, features and test modules;
• test cases each verifying a feature of a component, with all the commands needed to run the test, and how to setup the test environment;
• test reports which present the results of the execution of the individual tests.

4.6 Project management documents

All the tasks of the project are planned and tracked by the project management activities (see “Project Management and Tracking” in these proceedings). These documents should at least contain:

• a project plan, with the tasks description, resource allocations and dependencies;
• project scheduling and tracking documents, with progress status, Gantt and Milestone Trend charts (see [4], pg. 328);
• regular project reports, both for the team members and for the project customers.

4.7 Configuration management documents

The configuration management documents describe all the configuration items of the project; every artifact (document) of the project phases and activities (requirements, design, code, tests, etc.). They also describe the repository where all these documents are stored, how they can be modified, how they are versioned and released. See more details in “Configuration Management” in these proceedings.

4.8 Project infrastructure document

The project infrastructure documents are a set of documents that are different for different roles in the project (development infrastructure, management infrastructure, documentation infrastructure and so on). They basically describe how to participate in the project in a given role; therefore they contain information such as:

• all the conventions of the projects;
• which tools to use in the project and how they are used;
• templates and methodologies to follow;
• recommended reading for the project members;
• how to use the configuration system;
• how to report progress to the project manager;
• any other practical needs within the project.

5. USER SOFTWARE DOCUMENTATION

User documentation is not necessarily made of paper, it is all the information that is presented to the user: manuals, system or error messages, on-line help, Web sites, comments in the code examples, multi-media tutorials, etc. The appropriate tools should be used, such as word processors, Web builders, screen capture-replay tools and tutorial builders.

![Fig. 7 User’s and engineer’s views of a product.](image-url)
When producing user documentation you must always think of the user’s point of view (Fig. 7). It is not enough to build a good product, it is also important to have an adequate documentation: your product will never get a second chance to make the first impression! User documentation must not become a prose exercise, it should have the maximum amount of information with the minimum amount to read; the documentation is an important deliverable and it is developed in the same way as the application, with phases of analysis, design, editing, testing, etc. Documentation must be reviewed and tested with a few users, suggestions collected and new versions of the documents produced (and the changes should be highlighted).

There are a few immediate simple criteria to judge the quality of user documentation:
- bad documentation: engineer-centred, inappropriate layout and fonts, too wordy and verbose, too many cross-references and an inadequate physical format;
- good documentation: task oriented and user-centred, different formats and fonts for different purposes and user levels, allowing random access via indices and table of contents, complete textual information with a lot of graphs, pictures, etc.

6. SUGGESTIONS ON SOFTWARE DOCUMENTATION

6.1 Elements of good documentation

Try to convey information in a form other than text and prose use elements like:
- structured writing (with heading, small blocks, lists, sequences, etc.);
- flowcharts to show graphically the structure and the behavior of the product;
- trees and graphs to show the structure of the data
- lots of examples and playscripts that the user can use as such and understand how to proceed in practice;
- plenty of pictures and screen dumps to show, for instance, how to use the graphic interfaces.

6.2 The STOP method

![Fig. 8 The STOP method.](image)
In the IPT group at CERN we have written many documents using the STOP method and have experienced how positively the user welcomes them. This method was invented at Hughes Aircraft Company (1960) and it is widely used in several companies both for internal and public documentation (NCR, Texas Instruments, IBM and others).

This method has several advantages for the reader and for the writer. The principles of the method are simple:

- break the information you want to convey into two-page components;
- one concept per component;
- text on the left (500 words) and graphics on the right, both explaining the same concept.

In fact every time the reader turns the page he disconnects and reconnects to the book; “normal” books very often interrupt the reader’s attention even in the middle of an explanation. With this method, every time a page is turned a concept is completed and a new one arrives. Text and graphics are using different parts of our brain and some readers prefer one of the two types of presentation; by providing both forms we satisfy all kinds of audience.

For the editor it is also much easier to produce and reuse the material because the document is extremely modular, the graphic pages can be used as viewgraphs for presentations and the text is the track for the verbal part of the presentation.

7. CONCLUSIONS

I hope that in this summary I managed to convey the most important message I wanted to pass to you: if you document your work, you show that you are a mature engineer and a person who cares for colleagues and for the project. Believe me: there is really no point in working like mad and being able to explain and document only a fraction of the work you do.

Sometimes it is not our fault if we do not write documentation, but of the organization for whom we work: it is really a symptom of an organization’s immaturity to look at the amount of work its staff does without considering the quality of the documentation that is a part of that work!

So write documentation while you work, build (or look for) your document templates and use them consistently; in this way you will build a set of coherent documents; they are part of the product you are working on. Yes, they are a pain in the neck, but prevent greater troubles to you and to your colleagues.

8. RECOMMENDED READING LIST

8.1 Documentation in general


8.2 Documentation for software development


8.3 Software documentation standards


QUALITY THROUGH SOFTWARE METRICS

Arash Khodabandeh
Information and Programming Technology Group
CERN, Geneva, Switzerland

We have seen in the “Software Evolution - Track Introduction” that the CMM model evaluates the maturity of a software process. It defines 5 levels of maturity and the KPAs to be in place in order to reach any of those levels. This track is build around the level 2 and this talk concentrates on the Quality Assurance KPA.

1 SOFTWARE METRICS

The goal of Quality Assurance is to verify process and product compliance and to address non compliance. This verification can only take place if effective measurements are performed to quantify compliance. Measurement can affect all aspects of the software process including the software itself. Among various aspects of software metrics we will mainly concentrate on metrics derived directly from the source code.

The correlation of software metrics and source code quality have been verified by experimental studies conducted for example by T. McCabe and M. Halstead. Closer to us, E. Lancon studied the evolution of the ALEPH reconstruction code called JULIA from 1990 to 1995. It showed that the routines with the poorest quality in terms of metrics score are also the one modified most for bug correction; Moreover, after 5 years of such corrections, the quality of the code automatically improved.

Early measurement helps achieving quality by monitoring complexity, quantifying test effort, forecasting maintenance cost, identifying risks and planning preventive maintenance. Once the problem to be solved has been clearly defined, a quality model can be build combining the right metrics as basic components for measurement. The quality model defines the acceptable range of complexity where it is safe to be. It needs to be adapted both to the development team expertise and to the type of project.

Software tools can help collecting the metrics and assess the software compared to the defined quality model. The interpretation of the results remains a human activity, usually performed by the quality manager. One approach is to use source code metrics patterns to cope with the amount of
collected data. Although patterns are results of empirical studies and should be used with care, 6 of them were presented to be used during the lab work.

2 SOFTWARE METRICS LAB WORK

In the hands-on sessions, students practiced with Logiscope, a tool for software metrics. Logiscope is a toolbox for improving programming quality and test coverage. It can analyze more than 80 language variations including C, C++, Fortran and soon Java. Logiscope features include:

- Code quality with support for software metrics computation to assess maintainability, testability and component re-usability;
- Test coverage with support for coverage rates on source code branches, procedure calls, instruction blocks etc.;
- Code standards with support for verification of the program against programming rules and customization of rules to check;
- Graphical reverse engineering.

The lab works started with a walk through demo on a sample C++ code with step by step instructions through a possible quality analysis session with introductions to the most common features of the tool. For the remaining part of the session, students could analyze their own code (C, C++ or Fortran) in groups of 2 using provided list of hints, a road map and patterns presented during the lecture. A separate tool (J-Metric) was provided to students with Java code.

26 students have contributed with their own code from various experiments; 5 extra codes have been provided by the lecturers of the track. The range went from very small code to large codes such as LIGHT, OPAL trigger software of GEANT 4.

3 LAB WORK WRAP UP

The lab work were followed by a wrap up session where 11 groups presented very interesting results and their conclusions about the approach.

- Two presentations were followed by debates on “possible limitations of metrics restricted to source code only” and “the minimum size of the project (in term of LOC and people) were the approach could be used”.
• One group, who had problems with commercial Java codes, showed how metrics could have been used to monitor this complexity.

• Other groups explained how, as a result of the quality analysis, they undertook corrective actions to improve the quality of their code. They took the opportunity to stress the potential risk of micro-quality assurance versus overall quality assurance. Some took the results back home to share with colleagues.

• Last, two groups proposed a new metric and a new pattern reproduced in the appendix below.

4 REFERENCES


A new number:
The Ponchamatic number

The Ponchamatic number $PN$ is defined as the number of Ponchas the programmer drinks.

It must be comprised between 2 and 5 to achieve a good code. If less than 2 it produces boring programs, if higher than 5 the result is like this:
CONFIGURATION MANAGEMENT

Bob Jacobsen  
University of California, Berkeley

Doug Johnson  
University of Colorado, Boulder

Abstract  
The size and complexity of software projects in high energy physics has been increasing, resulting in increasing need for techniques to manage the software produced. Configuration management tools and techniques that can help with this problem are discussed here.

1. INTRODUCTION  
Configuration management, as implemented through a software release process, is the heartbeat of a large software development project. It provides the consistency and focus that is needed to keep such a project on track, developing usable software.

In this note, the second section starts by characterizing the problems encountered while building large software systems. The third section surveys existing practices both in high energy physics, and outside. The fourth section describes a particular set of procedures and tools used by the BaBar collaboration to do configuration management.

2. MANAGING DEVELOPMENT OF LARGE SYSTEMS  
The conventional wisdom is that large software development projects are difficult to control[1][2][3]. There are a number of factors that contribute to this:

1 Big projects have lots of code.
   The current generation of general purpose high energy physics experiments depends heavily on software for the operation of the detector and analysis of the data. It is no longer unusual for the on-line or off-line system of such a detector to have over a million lines of software in operation.

2 Big projects have lots of changes.
   Particularly during the development phase, the source code of the project is continually changing. The software for a large experiment might be developed over two or three years, which means that on average at least 5% of code changes every month. Due to updates, rework, migrations, temporary test scaffolding, etc., the observed change rate is often much higher than this. It is very important that none of these changes gets lost, and that all concerned are working with consistent copies of the code.

3 Big projects involve lots of people, and are often geographically distributed.
   Almost all high energy physicists have some involvement with software. Some work primarily on software, and some encounter it only peripherally in the course of an analysis project. In a large collaboration, its common for hundreds of people from fifty or more locations to be contributing to the development effort. Arranging for these efforts to add constructively, instead of interfere destructively, is clearly important.

4 Big projects are hard to understand.
   In spite of everyone’s best efforts, no system design is perfect and complete. As the size of the system grows, so does the number of interfaces that must be specified, the number of users of each
interface, and the number of unforeseen special cases that a typical interface must satisfy. In effect, the complexity of the system grows much faster than its size, and usually much faster than its capabilities. It is often very difficult to tell what software will be effected by a change, or whether two given pieces will successfully interact.

“Configuration management” refers to the activities necessary to control the actual products of the development effort, including source code and the resulting libraries and programs. One specific definition is:

“The purpose of Software Configuration Management (SCM) is to establish and maintain the integrity of the products of the software project throughout the project’s software life cycle. Software Configuration Management involves identifying the configuration of the software (i.e., selected software work products and their descriptions) at given points in time, systematically controlling changes to the configuration, and maintaining the integrity and traceability of the configuration throughout the software life cycle.”[4]

At first glance, most of these problems appear to be addressable at the level of source code. For example, if two developers can agree that they each have an identical set of source code for the entire project, they can then usefully discuss changes that they want to make cooperatively. A sufficiently powerful “source control system” should make this possible.

Unfortunately, this is not enough. It is also necessary that each of the developers can build an equivalent system from their copy of the code. Without that, it is not possible for them to work cooperatively on parts of the project, and it will be very difficult to merge their efforts into a single working system. In other words, the procedures and tools for building the actual system from the source must also be managed if people are to be able to work effectively together.

Further, it is difficult to control large projects at the level of the source code. It is not practical to give each developer a complete copy of the source code; nobody wants to compile all that for themselves! Operating systems provide libraries to get around this, with the assumption that once created, a library can be used by many people. Sets of libraries, along with consistent header files and documentation, are often called “releases”. Unfortunately, most operating systems do not provide facilities for version control of libraries, determining which source generated the library, and ensuring that compatible header files are used with the library. All of those must be provided by a separate “release control system”.

Finally, it is often difficult to tell whether the software project as a whole is progressing. There is much anecdotal evidence for projects that stayed “90% complete” for months or years, and then finally failed. Many of these are blamed on problems with integration; the vast bulk of the code had been written in parallel, and it turned out to be impossible to connect those pieces so as to have them work together. One effective technique for addressing this is “cyclical development”, where partially-functioning versions of the system are repeatedly built and used as development proceeds. Perhaps the limiting case of this is the “nightly build”, where the entire set of changes are integrated and tested every 24 hours. A system for identifying the correct contents of such system builds, coupled with a means of effectively testing it, is a vital tool for development projects of duration longer than the average physicist’s attention span.

“Study it forever and you’ll still wonder. Fly it once and you’ll know.”
- Henry Spencer

3. EXISTING PRACTICES

This section is a brief survey of some existing configuration management systems in use by the high energy physics community, followed by a longer discussion of the recommendations of the Capability Maturity Model.

3.1 Systems Traditionally Used in High Energy Physics

There are a number of source control systems long in use in high energy physics[5]. On VAX computers, perhaps the most popular ones are “CMS” and “CMZ”. “Historian”, “Revision Control
System” (RCS) and home-brewed systems are also used. Most of these control the contents of individual files or directories, and allow the user to label specific versions of the contents with “tags”. For example, the tag “V01” might refer to a specific set of contents of a file, with “V02” referring to a later version. This ability to label file contents is most useful when it can be applied to a large set of files. It can then be used to express concepts like “Tag V23 refers to the source code for this executable”; “The on-line system was built from tag V84”, etc. Before changes are made, a file or directory must typically be “reserved”, providing an exclusive lock for a single developer. Once the changes have been made permanent, this lock can be released. This serialization can result in significant interference between developers.

For traditional FORTRAN systems, only minor infrastructure has been needed to ensure consistent executables are built from the source. Typically, experiments would develop small scripts to handle the compilation and linking of the entire set of code. As more use is made of C-type languages, particularly C++, this has become somewhat problematic. This family of languages uses large numbers of “header files” to declare interfaces to classes and functions in libraries; these must be consistent with the actual library contents to avoid subtle and hideous errors. To handle this and to ensure that everything necessary is recompiled after a source change, many experiments have adopted the Unix “make” utility, along with providing a set of the necessary “makefiles” to control it. “make” uses knowledge of which compilation units depend on which source files, so that it can recompile anything that might have changed. Coding the needed rules to configure this is difficult, however, and there are only limited debugging facilities available.

Many experiments address the need for a release system by providing a human being with the title of “Librarian”. The librarian does the creation of libraries and executables, perhaps with the help of some automation, then installs them into publicly available locations. In some systems, such as the CERNLIB libraries, several copies are kept to allow a user to specify whether to use the “development”, “production” or “old” one. Typically only a few are available, and the migration of them is entirely beyond the control of the individual user. Tools like UPS and UPD from Fermilab can help with setup and distribution, but the workload of the librarian tends to be large. Because the human librarian is almost always overloaded, it is generally not possible to build and test a complete copy of the software very often, which results in the inevitable integration being generally quite difficult.

3.2 The Capability Maturity Model

The Software Evolution track[6] of this CERN School of Computing is based on a specific framework: The Capability Maturity Model (CMM)[4]. It has been developed by the Software Engineering Institute at the Carnegie Mellon University. Based on the most effective groups observed in industry and academia, their is a significant amount of empirical study underlying the CMM. It is gaining wide respect within the software community. The CMM does not address itself to particular tools or technologies; rather, it provides a framework for evaluating particular technical choices. In the specific area of configuration management, the CMM points out four specific goals (Table 1). It is difficult to argue with these.

<table>
<thead>
<tr>
<th>Table 1. Goals of the CMM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) Software configuration management activities are planned</td>
</tr>
<tr>
<td>2) Selected software work products are identified, controlled, and available</td>
</tr>
<tr>
<td>3) Changes to identified software work products are controlled</td>
</tr>
<tr>
<td>4) Affected groups and individuals are informed of the status and content of software baselines</td>
</tr>
</tbody>
</table>
To reach these goals, the CMM recommends that a software project have a “commitment to perform”:

The project follows a written organizational policy for implementing software configuration management (SCM).

It goes on to say that this typically specifies that:

- **Responsibility for SCM for each project is explicitly assigned.**
- **SCM is implemented throughout the project’s life cycle.**
- **SCM is implemented for externally deliverable software products, designated internal software work products, and designated support tools used inside the project.**
- **The project establish or have access to a repository for storing configuration items/units and the associated SCM records.**

Once that commitment is agreed to, the CMM proceeds to recommend ten specific activities to perform (Table 2). It would be difficult to object to any of these; they all seem quite necessary on the face of it. But the CMM does not provide any real guidance on how to actually carry these out, nor on the level of detail needed in each of the activities; it is expected that each project will adapt these points to their specific needs.

<table>
<thead>
<tr>
<th>Table 2. CMM Activities to Perform</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) A SCM plan is prepared for each software project according to a documented procedure</td>
</tr>
<tr>
<td>2) A documented and approved SCM plan is used as the basis for performing the SCM activities</td>
</tr>
<tr>
<td>3) A configuration management library system is established as a repository for the software baselines</td>
</tr>
<tr>
<td>4) The software work products to be placed under configuration management are identified</td>
</tr>
<tr>
<td>5) Changes requests and problem reports for all configuration items/units are initiated, recorded, reviewed, approved, and tracked according to a documented procedure</td>
</tr>
<tr>
<td>6) Changes to baselines are controlled according to a documented procedure</td>
</tr>
<tr>
<td>7) Products from the software baseline library are created and their release is controlled according to a documented procedure</td>
</tr>
<tr>
<td>8) The status of configuration items/units is recorded according to a documented procedure</td>
</tr>
<tr>
<td>9) Standard reports documenting the SCM activities and the contents of the software baseline are developed and made available to affected groups and individuals</td>
</tr>
<tr>
<td>10) Software baseline audits are conducted according to a documented procedure</td>
</tr>
</tbody>
</table>

Finally, to manage and improve the software configuration effort, the CMM recommends five particular actions be taken under “measurement, analysis and verification” (Table 3). Again, these are generally not controversial. The most common significant objection seems to be “we don’t have time or people to do all these”, and that’s often true. But the authors of the CMM argue that most of the gain is from the
first small amount of effort, and maintaining a consistent allocation of effort in this area, even if small, is worthwhile.

Table 3. CMM Measurement, Analysis and Verification

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Measurements are made and used to determine the status of the SCM activities</td>
</tr>
<tr>
<td>2</td>
<td>The SCM activities are reviewed with senior management on a periodic basis</td>
</tr>
<tr>
<td>3</td>
<td>The SCM activities are reviewed with the project manager on both a periodic and</td>
</tr>
<tr>
<td></td>
<td>event-driven basis</td>
</tr>
<tr>
<td>4</td>
<td>The SCM group periodically audits software baselines to verify that they conform to</td>
</tr>
<tr>
<td></td>
<td>the documentation that defines them</td>
</tr>
<tr>
<td>5</td>
<td>The software quality assurance group reviews and/or audits the activities and work</td>
</tr>
<tr>
<td></td>
<td>products for SCM and reports the results</td>
</tr>
</tbody>
</table>

The aim of the initial levels of the Capability Maturity Model is “repeatability”. They argue that it is difficult, perhaps impossible, to improve the way that software is built until it is possible to reproduce those methods.

But remember, the goal is repeatable success. A repeatable failure is still a failure. And systems that are defined, perhaps after great effort, but then are not used by the developers are worthless.

“Producing a document like this also entails a risk. Some individuals will always seek fixed formulas or checklists for evaluating organizations. When organizations follow this checklist strategy, they often produce piles of documents and mountains of paper to “prove” that their process is at some prescribed level. Unfortunately, this approach invariably overlooks the critical point: what people actually do. The software process concerns people and their work and must be pragmatic and adjustable, or it will not be used.”

- from the CMM foreword

4. AN EXAMPLE OF CONFIGURATION MANAGEMENT IN HEP

The BaBar experiment, described in detail elsewhere[7], will start taking data in early 1999. It is developing its off-line software primarily in C++, with a small mixture of FORTRAN and other languages. With the BaBar off-line project, there are four major components to the adopted configuration management solution[8][9]:

1 Establish responsibility:

   The BaBar off-line code is divided into approximately 200 “packages”, each with a “package coordinator” who is a specific individual who has agreed to the job. The coordinator of each package is responsible for the contents of the package, in the sense that the coordinator must know the status of the package, and must determine when a new version is ready for use by the collaboration.

2 Control the integrity of the source code

   BaBar uses the “Concurrent Versioning System” (CVS)[10] to provide versioning and control of source code. CVS is a publicly available source management system that does not require users to obtain an exclusive lock on files before starting to modify them. This, along with other CVS capabilities described later, makes it a much more manageable tool for a large project.

3 Drive the integration with “releases”
The BaBar collaboration has developed “software release tools” (SoftRelTools or SRT)[11] for creating consistent releases of libraries and executables from specified versions of the source code packages. These releases can then be distributed to remote sites, checked for integration problems, used as a base for further development by individuals or groups, etc.

4 Make this the heartbeat of the project

On a regular basis, currently every two weeks, the entire BaBar off-line system is built as a new release and lightly tested. This periodic integration and testing provides a basic timescale for development efforts: developers can plan their work so as to get it included in a specific one of these periodic releases. Due to the large number of packages changed during one of these development intervals, and the size of the releases, it is important that the release process be sufficiently automated.

The next four subsections describe each of those points in more detail.

4.1 Establish responsibility - Packages and Package Coordinators

Large projects need to be subdivided into reasonable-sized units. These then become the basic unit of testing and release. Within the OO paradigm, these are often called “class categories”. Within BaBar, these are called “packages”.

The basic approach is simple: Find a person to take responsibility for each package, called the package coordinator, and then build a support structure so that the package coordinators can concentrate on developing their packages.[12]

The partitioning of the system into packages is important for many reasons. Some of these are technical: to foster design independence, to allow implementation to proceed in phases, etc. Just as important, however, are managerial issues. Package lines need to take into account organizational lines, and the preferences of the people serving as package coordinators. For example, the software specific to a particular hardware system’s simulation and reconstruction is the responsibility of the hardware construction group, so the package boundaries need to reflect this. All of the following have been used to decide where to separate BaBar code into distinct packages:

- Separating interfaces from implementations
- Separating persistent code (e.g. code associated with a particular database implementation) from the corresponding transient packages
- Separating graphics code from the underlying computational classes
- Separating out code of common utility
- Separating code used in the reconstruction of real data from that used for simulating events, from that used for analysis of the resulting data
- Separating algorithmic code from calibration and alignment code
- Separating code associated with different hardware systems
- Defining a set of code to be produced and maintained by a particular institution
- Inability of two developers to effectively collaborate

Over time, the set of packages evolves. Common code is identified within the system specific packages, and migrated to new common packages. New general categories such as “geometry” or “calibration” are introduced, and packages are generated to contain that type of code within the specific detector hardware systems. Its important that the source management and release management structures of the project make these migrations practical. BaBar does this through various technical procedures that permit changes to be “flushed through the system” over the course of several biweekly releases.

The package coordinators have a critical role. Although they are generally the most active developers of a particular package, they need not be. Rather, they are responsible for managing the integration of their package within the overall software project. At the most detailed level, this means...
that they have to resolve problems that are found, either by fixing them directly or ensuring that somebody else does. They are also responsible for the documentation updates, implementing any changes needed by migrations of other code from package to package, for keeping test scaffolding up to date, and for the other ongoing maintenance that’s needed to keep the package working during development of the overall system. At the largest level, its the package coordinators responsibility to plan out the long-term evolution of the package, including how it fits into the overall system and what new capabilities will be added.

Currently, the BaBar off-line project contains about 200 packages, with 95 separate package coordinators. Packages are typically 3,000 to 30,000 lines of code, with most around 2,000 to 4,000. The largest are legacy FORTRAN packages; the smallest define abstract interface classes, without associated procedural code. Approximately 40% of the packages are changed during the course of a two week development cycle. There are generally two or three new off-line packages per cycle, typically due to splitting an existing package into two to reduce dependencies.

To make this work, a significant amount of automation is used. There are web forms for creating new packages, and for requesting that a specific version of a package be tested on each of the BaBar supported computer architectures and compilers. Flexible tools have been developed for managing the link lists and compilation dependencies. Automated documentation and indexing tools are being deployed.

4.2 Controlling the integrity of the source code - CVS

The CVS source code management system maintains a repository of text files, allowing users to check-in and check-out specific versions of changed text, and provides a facility for tagging particular sets of contents. Older versions of the files remain available forever, as the repository contains both the most recent contents and enough change information to be able recreate any specific older set of contents.

The primary advantage of CVS over similar systems is it does not use exclusive checkouts to manage concurrent development. More than one developer can have the same file checked out at the same time. This is quite important in a large system where the source code is arranged in packages of a finite size, as it is common for two developers to be working on different files in the same package. Conflicts between updates from two different developers are rare in practice, and CVS provides a comprehensive set of tools for detecting and handling them.

4.2.1 Examples of CVS usage

CVS is controlled by the single $CVSROOT environment variable:

```bash
setenv CVSROOT ~foo/repo
```

To get a copy of the most recent contents of a package Foo, the check-out command is used:

```bash
cvs checkout Foo
```

This is often referred to as the “HEAD” version of the package, as its considered to be at the head of the development path. To get a copy of a specific older version (tag), such as V00-02-23, of a package Foo:

```bash
cvs checkout -r V00-02-23 Foo
```

These checkout commands produce fully editable Foo directories, etc. The user can then edit, compile, link and test that software without worrying about changes that somebody else may make in the repository. As development proceeds, however, somebody else may put changes to the package back into the repository. If the user wants to see a list of which files have changed:

```bash
cvs -n update -A
```

The “-n” option to most CVS commands gives a summary of what actions the command would otherwise take, but suppresses the actual execution of them. The update command will list files that have changed in either the local copy or the repository, along with a single-letter flag that describes what action will be taken. To actually update a directory to the most recent contents:

```bash
cvs update -A
```

This will merge any new changes in the repository into the users current working directory. It is also possible to update to be consistent with a specific tagged version with the command
cvs update -r <tag>

Note that any changes the user has made in the current directory will not be placed in the repository by the update command; update only moves changes in one direction. Changes are placed in the repository with the “commit” command:

cvs commit

The commit command will prompt the user for a comment to be entered in a log file, and optionally to be emailed to people who have registered their interest in changes to this package. The commit command can also take many options, allowing the user to commit only one file at a time, control processing of subdirectories, etc. The commit command can also fail. This is usually caused by CVS being unable to get a lock on the repository; although the checkout procedure does not lock files while they are being used by somebody (typically days or weeks), it is necessary for CVS to lock the repository during the time a checkout or commit command is actually running (typically seconds) to prevent repository contents from changing while they are being used. CVS will wait a short while to attempt to get the lock and then quit with a notification to the user; a retry will often be successful. A commit can also fail due to an unresolved conflict with the contents of the repository. This typically happens when another user has already committed some changes to the package, and these have not been merged into the current working directory with the “cvs update” command. CVS does the merging and resolution of concurrent changes during the update, not during the commit.

To keep track of changes, CVS uses a specific “CVS” subdirectory within each checked-out package. This contains files that CVS maintains; they should not be touched by the user. Within it, CVS records which version numbers of each file the user has checked-out to the local working directory. When its time to update a file, CVS compares the version originally checked-out to the current contents of the working directory; this comparison will show any changes made by the user. CVS also compares the version originally checked-out with the version to which its trying to update; this will show any changes made to the repository since the checkout. If those two sets of changes do not overlap, which is the usual case, they are considered to not conflict, and the second set of changes can be applied to the working directory.

If the changes do overlap, a “conflict” has occurred. CVS cannot resolve this, as it requires some human to understand the changes and merge them. CVS gives the user a message that this has occurred, and marks the conflicts within the file with:

```
<<<<<<<<<
One content

Other content
```

The user must take an editor and correct these. Often that’s as simple as recognizing that both changes do the same function, and just selecting one. Sometimes both changes are needed, and the only decision is which order to put them in. Occasionally, there is a real conflict which requires thought to correct. Its rare for codevelopers to really conflict by changing the same line, because usually there’s only one person working on a particular piece of functionality. The most likely cause of a real conflict is a migration from one form of an interface to another, which can sometimes require sweeping through large parts of the project changing the usage of the interface. If developers are working too far behind the HEAD version, e.g. with code that is too far out of date, they will sometimes encounter conflicts as these migrations are added to the HEAD.

4.2.2 BaBar extensions and usage

BaBar uses a customized version of the CVS program to add additional logging and protect against the misuse of certain options. As there is currently almost 3 million lines of text in the BaBar repository, it is considered a critical resource and some effort is made to protect it with Andrew File System (AFS) access control lists (ACLs) and backups.
BaBar has also implemented an “expert list” for each package. People on these lists are notified when the package is modified. The email notification contains which files were changed/added/deleted, who did it, when, and the text of the comment they entered. Package coordinators use this facility to stay abreast of changes to their packages. Other developers also find it useful to get advance notification of changes to packages that their code depends on.

Note that any BaBar member is technically permitted to commit changes to any off-line software package. When people encounter and track down a bug, they are encouraged to commit the fix for it directly to the CVS repository, rather than describing that fix to the package coordinator. The vast majority of commits of this type are correct and useful, and the effort saved by not having to work through multiple people to make a minor change more than offsets the small amount of corrective effort spent on bad changes. This relies heavily on the package coordinators, however, as they must stay abreast when CVS notifies them of the changes being made to “their” packages.

During the summer of 1998, approximately a year before the turn-on of the BaBar detector, there were about 450 commits to CVS during a typical month. As more than one file can be committed at a time, this corresponded to about 700 files changed per month, about 15% of the repository. Approximately a third of these modifications were modifications to existing code, with the rest being new functionality of one type or another. Approximately half of the modifications have come from computers remote to the Stanford Linear Accelerator Center, which hosts the repository. CVS has generally functioned well for BaBar, and seems to be able to scale upward even from this level of collaboration-wide usage. There are also a small number of people within BaBar who have created their own, private repositories and use CVS to maintain a history of versions of their own private work. It is remarkable that CVS can work well across this entire spectrum of uses.

4.3 Drive the integration with releases - SoftRelTools

A large project needs to keep people working from the same code base, so as not to allow too much divergence, and keep the integration effort affordable. A plan like “Take the HEAD of CVS for everything and compile it” does not address this, as “HEAD” changes over time. Putting a single tag on every file in the CVS repository would provide consistency, but would also require an unrealistically large amount of communication and coordination to among the complete set of package coordinators, effectively making this an intractable \( O(N^N) \) problem.

A release system is meant to reduce this load by spreading the communication out to the package coordinators. Each package coordinator makes a determination of the specific version of their package(s) that will be included when the next release is built. This may involve coordination with the package coordinators of a few other packages, due to inter-package dependencies, but at least not every package coordinator has to know the most recent changes to every package. The entire release is then built from this set of specified versions, and tested to make sure no problems have crept in due to misunderstandings on the part of the package coordinators.

This system also serves to decouple when a change is committed to the repository, and when it is used by the larger development community. A specific change to a file is not used by general developers until it appears in a release, and that doesn’t happen until the relevant package coordinator has decided that it is ready to be used. This decoupling permits developers to collaborate by working with the contents of CVS, rather than forcing experimental code out into the entire collaboration. This is particularly important during the parallel development phases of a project, during which it is necessary to keep a (partially) working version of the system available at all times.

The release system is also meant to make development easier. To this end, it provides “test releases”, in which a developer can check-out a few packages, modify them as needed, recompile, relink and test. Packages that the developer need not modify, but that are needed for the executables being tested, are taken unchanged from an underlying production release. Each developer can have as many different test releases as desired, allowing separation of things that are being done on different timescales or with different people. This also allows an individuals work to proceed independently from the overall release schedule; so long as a specific production release remains disk-resident, it can be
used as a base for test releases. Developers need not move their work forward every time a new production release is created.

Finally, the release system has to allow simultaneous development on multiple machine architectures, known as “platforms”. Over time, BaBar has supported nine different vendor’s equipment, with up to five active at a single time. Currently only two are supported, but that is expected to rise to four in the next year or so. The release system supports multiple platforms by separating the source code, which is platform independent, from the libraries and binaries, which cannot be. AFS is used extensively to allow a single file tree to be used from multiple different machine types, and a user can issue the same “gmake all” command regardless of which supported machine is currently in use.

The BaBar release system, known as SoftRelTools (SRT), does all this through use of a specific directory structure shown in the following figure. More detail is available elsewhere.

4.4 Make this the heartbeat of the project - The automated release system

To reduce uncertainty about integrating the complete software system, and to provide a common direction to the overall development, it’s important that the entire system be built routinely. An infrastructure for doing this has been created on top of the capabilities SoftRelTools provides for building a single production release. It centers around a Release Coordinator, who is responsible for regularly building complete releases of the system.

Note that the Release Coordinator does not decide which versions of specific packages will be included in these releases; that is the role of the individual package coordinators. Rather, the Release Coordinator creates and uses automated procedures to gather together the tags specified by the package coordinators, build the resulting software system, and release it back to the package coordinators along with digests of the error messages encountered during the build and test procedures.

The BaBar release cycle is currently two weeks. At any point, package coordinators can fill out a web-based checklist to “announce” a new version of their packages. The checklist information is recorded, and a comment is entered in a searchable list of package announcements. During most nights, a “nightly build” is automatically run, which starts from the most recently announced versions, and

![Diagram of the SoftRelTools directory structure]

Fig. 1 The SoftRelTools directory structure
which ends by emailing any error messages encountered back to the relevant package coordinators. The nightly build is not intended as a base for ongoing development, and the resulting libraries are not kept around past the start of the next nightly build. Rather, it is meant to provide early warning of any integration problems that might inadvertently result from a change to a package.

On alternate Tuesday evenings, a more permanent production build is made. This one is intended to be a basis for development, and package coordinators are strongly encouraged to announce their new versions enough days in advance that the Tuesday build is likely to compile and link cleanly. Our Tuesday builds currently compile over 2 million lines of code, and link more than 140 test executables, taking about 16 hours. Problems encountered in a Tuesday production build are meant to be addressed in the Wednesday nightly, and an additional “production rebuild” is generally done on Thursdays containing only specific corrections intended to get a clean result. This is then kept around for at least a couple of months as a basis for ongoing development.

Note that, with some exceptions, this cycle runs regardless of whether some particular package is ready or not. Development efforts that fall behind will miss inclusion in a particular Tuesday’s production build, and will have to be included in a later one. The alternative, delaying these common release builds until “everybody’s ready”, has been found to result in large overall delays. After all, some part of a project is always late; if the parts that are on schedule at any given moment are forced to wait for those that are behind schedule, pretty soon everything will be behind schedule.

A remarkable amount of automation and effort by the Release Coordinator is required to make this work. The package coordinators need to have the system be foolproof and have minimal impact on their effectiveness. Ideally, they spend a minute or so filling out a web form, and the next day will get email showing the results of including their work in the complete software system. They also need web pages that can show them the complete set of packages that have been announced since the last release, allow them to easily locate the package coordinator for a particular package, and request test compilation and links in between the nightly builds. The underlying mechanisms of the build system are necessarily spread across multiple computers of different types, and susceptible to the usual vagaries of network outages, disk errors and AFS token expirations.

The release system also produces statistics for monitoring problem rates, and is interfaced to the “Remedy” system BaBar uses for tracking problems. Planned future enhancements include adding automated generation of various metrics and code quality checks to the biweekly production releases.

5. CONCLUSIONS

Managing the logistics of large software development projects has historically been difficult; we seem to always bite off more than we can chew. The methodologies of the Capability Maturity Model, tools such as CVS and SoftRelTools, and the techniques described here can help with this. They have been tested in the BaBar off-line system, a project that has exceeded one million lines of code, and have been found to greatly help. Components of these have also been used with success on other large high energy physics projects[13][14].

Acknowledgments

We wish to thank the Sloan Foundation for their generous support. Much of the technology described here was developed at the BaBar collaboration. Ed Frank, and Terry Hung have been particularly important to the fine-tuning of these ideas as applied to BaBar. Liz Sexton-Kennedy and Robert Harris of CDF have also provided valuable feedback.

References


[5] An excellent discussion of these issues as seen from FNAL is in CDF Note: CDF/DOC/COMP_UPG/PUBLIC/3891; D0 Note: 3118d; Fermilab Computing Division Note: GU0013, available at http://www-cdf.fnal.gov/offline/code_management/run2_cmgt/run2_cmgt.html


[10] CVS documentation is available at a number of Web sites, including http://hill.ucp.ualberta.ca/Documentation/cvs-1.8.1/cvs_toc.html


SOFTWARE EVOLUTION: CASE STUDY, OPAL

K. Ackerstaff
CERN, Geneva, Switzerland

Abstract
The OPAL case study presented in two lectures shows the application of modern software engineering techniques presented in the lectures on user requirements, project management, configuration management, software documentation and software quality in the Software Evolution Track in a running high energy physics experiment.

1. INTRODUCTION
In HEP (High Energy Physics) the use of computers is no longer restricted to numerical calculation of complex mathematical problems. Today computers and software are used in all areas of HEP experiments, from administration to physics analysis. The size and complexity of current and future HEP projects makes computers and software a major tool to perform fundamental physics research. Software has become a “mission critical” part of HEP and has to be treated with a professional approach. With the advent of large scale experiments involving more than 300 people, huge apparatus, a major financial investment and a lifetime of more than a decade, the management of people, money, hardware and software engineering becomes a problem which can only be solved using the expertise of professionals. Feedback from Software Engineering Science into HEP is essential to cope with problems in software evolution in HEP.

The application of modern software engineering technology and tools is not very widespread in current HEP experiments. The tradition, or culture, of software development in physics departments of universities and institutes is one reason for this. Traditionally software for physics has been produced by physicists and many of the foundations of software engineering actually stem from physics research. Meanwhile the use of computers and software outside research has increased dramatically and software produced for HEP is only a tiny fraction of the total. The introduction of software engineering methodology and tools into the HEP community is very slow. Since many of the developers are not software engineers, new methods need training and the reluctance of people to change their way of producing software must be overcome. The culture of software development in HEP, the “way we do things”, can only be changed gradually since the people involved mostly spend only a fraction of their time on software development and come from different institutions from all over the world. “A new way of doing things” can not just be imposed on developers, but the challenges of yet bigger collaborations and experiments, with more and more complex software and an increased dependency of the final outcome of HEP experiments on this software requires the use of modern software technology.

In this case study the introduction of modern software development techniques into a running HEP experiment is described.

2. THE OPAL CASE
The OPAL experiment is one of the four experiments at the LEP particle accelerator at CERN which has been in operation since 1989. Amongst other software and hardware upgrades, the online system was upgraded and partly redesigned in the period 1996-98. With the advent of the LEP2 era in 1996, the accelerator is being continuously upgraded to gradually reach a maximum centre-of-mass energy of approximately 200 GeV. The four experiments, which detect particle collisions at LEP, must adapt to the changing accelerator conditions. The OPAL experiment has experienced substantial increases in background. The planned increase in both the beam energy and intensity is expected to degrade background conditions further.
The OPAL trigger and data acquisition system must be able to handle this background without losing physics events. The original system contains considerable flexibility in programming the fast logic used in trigger decisions. However, during the (long) lifetime of the experiment the trigger control software has had to adapt to changes in the detector, the trigger hardware and the accelerator operating modes, which were not foreseen in the original design. This resulted in a control code which was excessively complex and difficult to maintain.

The difficulties experienced in adapting the trigger control software to these unforeseen changes motivated the redesign of the software. In turn, the positive experience from this redesign led to an upgrade of the event builder involving a redesign of all concerned software.

This case study is restricted to the trigger upgrade which enables to show all aspects of software evolution and use of modern software technology in OPAL. The redesign and implementation of the OPAL trigger software was carried out by a small team from the OPAL experiment in close collaboration with one member of the CERN/ECP Information and Programming Technology (IPT) group. The adoption of suitable methodologies ensured the successful outcome of the project, despite the heterogeneous and real-time environment and the strict time constraints of a running experiment.

3. THE OPAL TRIGGER

The OPAL trigger system [1] selects events according to logic implemented in several custom made VME compatible modules. Programmable combinations of 120 trigger signals from 32 subdetectors are used to form the trigger decision, made every 22 $\mu$s. A positive trigger decision is sent on a dedicated bus to the subdetector data acquisition systems. After complete readout the subdetector and trigger data are collected by the event builder. The complete events are sent through the filter [2] to data storage. The entire data acquisition system is controlled by RunControl using finite state machines (Further details of the OPAL data acquisition system are given in [3]).

The original software ran on two VME CPUs and controlled and monitored the trigger hardware and signals. It was written in real-time Fortran [4], consisted of approximately 60000 lines, and was implemented under the OS9 operating system.

The main tasks of the trigger software are:

– to load the user defined trigger logic into the hardware;
– to read out the trigger hardware on each event upon a positive trigger decision;
– to trigger and synchronize subdetector readout;
– to read out monitoring information asynchronously to the event loop and send the data across the network;
– to handle exceptions generated by hardware and software.

The user interface to the trigger monitoring was a commercial histogram presenter which ran on a Macintosh II [5] connected via a VME interface card to the trigger crate.

4. THE REDESIGN

A team of six people worked full- or part-time on the redesign of the trigger software, in a project which lasted from January until August 1997. The project was led by a physicist and supervised by the OPAL online coordinators. The redesign was split into three sub-projects: the core trigger code; the histogramming presenter; the trigger monitoring software. The redesign of the core trigger code is used here as an example to demonstrate the importance of the approach to software engineering.

4.1 Motivation

The experience of running the trigger system in 1996 when LEP2 started motivated the decision to redesign the software and streamline the hardware. The main motivations were to increase maintainability and flexibility.
Several hardware upgrades carried out in the past could not be fitted into the original design of the old code, which resulted in excessive code complexity. The documentation did not reflect many of these code changes. Further hardware upgrades are foreseen during the remaining lifetime of the experiment, and the manpower required for operations must be reduced. Maintainability can be improved by moving the non real-time functionality to a UNIX platform and thus simplifying the VME hardware. Moving the histogram presentation to the common UNIX platform also removes dependence on specific hardware, and reduces the number of operating systems used.

4.2 Requirements & Constraints
The new system should cover the full functionality of the old trigger system, and satisfy additional requirements in order to implement the improvements mentioned above.

– The core trigger and the monitoring code have to be separated. The remaining core trigger code has to run on a single VME CPU with no loss of performance.
– The monitoring code has to be moved outside the real-time system.
– The histogram presenter has to be replaced and run on a Unix platform.
– All software must be documented, and designed to be maintainable.
– The design has to facilitate future hardware changes.

To achieve this an object oriented approach was chosen, using the C++ programming language. The redesign of a central software system within a running experiment imposes several major constraints.

– The existing interfaces between the trigger and outside software systems must be retained, for example databases, event structures and RunControl interfaces.
– The new system must run with the old trigger hardware.
– To ensure a smooth transition, the new system must initially be backward compatible to avoid any loss of data during commissioning.
– Finally the new trigger system must be complete and operational for 1997 data taking, using the manpower available.

4.3 The Software Development Process
The first step in the project is the choice of methodologies in the different activities and phases shown in figure 1.

The user requirement document (URD) and the project management follow ESA’s Software Engineering Standards (PSS05) insofar as was appropriate to the project.

For design and implementation the object modeling technique (OMT) was used.

Quality assurance was carried out by peer review and regular analysis of the code quality according to the ISO9126 [8] standards. The project was reviewed on a weekly basis in project meetings. The OPAL online and operations coordinators took part in the review process to ensure that requirements were met.

Fig. 1 The software life cycle used in this project. Activities such as documentation and management continue throughout the lifetime of the project, whereas project phases are active only during a specific time.
4.4 The Software Development Environment

In order to instrument the software development process a software development environment (SDE) was set up at the beginning of the project. Tools for each activity and phase of the process were selected and customized to the specific requirements of the project. Documentation is written using FrameMaker [9] with templates provided by the CERN/IPT group. These documents are hyperised using WebMaker [10] and linked to the OPAL online web. The source code is hyperised using LIGHT [11]. For configuration management the CVS [12] package is used. All relevant documents, utilities and source code are managed under the CVS repository, and changes to the repository are logged automatically on the web. CVSWeb [13] is used to browse the central repository via the web.

Code is developed on Unix workstations, whereas compilation, testing and operation take place on OS9 systems. For file and directory transfer across these platforms a tool has been written specifically for the project. The quality of the code has been analyzed along ISO9126 quality standards using Logiscope [14], which has been integrated to work with the GNU-make [15] program. Thus GNU-make allows compilation and linking on OS9 systems and automated quality analysis on Unix platforms using the same utility.

As an instrumentation of OMT, Rose [16] is used to produce the C++ class and inheritance as well as message trace and state diagrams.

The ATLAS coding conventions [17] were customized for the project. SNiFF+ [18] is used as the code development environment.

The knowledge and experience of the software engineer from the IPT group has been essential in setting up the SDE and choosing appropriate methodologies and tools. The project team benefited from this in terms of learning and applying the methodologies and tools. The setup of the SDE and the integration of the tools described above took approximately a month at the start of the project. Analysis, design and implementation took altogether three months, and the integration and testing phase two months. This time could only partly be used due to the constraints of the commissioning of the OPAL experiment and the LEP schedule, when the OPAL data acquisition system was often unavailable for testing the trigger.

4.5 Experience with the Software Development Environment

A detailed documentation of the project infrastructure includes a description of all technical actions in the software development process. This allowed all members of the project team to rapidly learn standard actions and adapt to the SDE. Therefore all tools could be used from the outset of the project. This documentation proved to be absolutely essential in order to get people to use and work within the SDE.

A substantial amount of information on the details of the trigger system, for example exception handling and interfaces, had to be retrieved from the old trigger code. SNiFF+ has been used to analyze the old software, and FrameMaker used to document the software and hardware, which facilitated the re-engineering process. A detailed quality analysis using Logiscope helped determine the critical parts of the trigger software. SNiFF+ also allows simultaneous navigation through both the old and new software during the analysis phase.

The OMT design enables fast and efficient coding. The models produced with Rose can be tested against the functionality of the old code. Hence the implementation of the models is separated from design and can be carried out by any member of the project team. Furthermore, inheritance allows partial testing. For example, the trigger hardware is reflected in hardware classes inherited from a VME module base class. This base class hides all VME specific memory access and error handling. This class allows either direct access to hardware registers, or else the registers are simulated in local memory if desired. This allows the software to be run on a test setup with only a subset of the hardware modules present. Hardware which cannot be accessed is automatically simulated in software.
The members of the project team were able to work simultaneously on the code using CVS. Following the coding conventions results in a uniform code structure which facilitates peer review and collaborative coding. The majority of version conflicts are resolved by the CVS merging feature. Version tracking and logging are important also during the implementation phase.

4.6 Product Assessment

Figure 2 shows the overall quality of the new code against time compared to the old code. The percentage of code rated as fair or poor according to the ISO9126 standards is below 2%, compared to above 30% for the old code. The improved code quality is maintained throughout the implementation and integration phases.

The improved quality is reflected in the performance of the new code. It has been operational since the beginning of August 1997, and satisfies all requirements. Unlike the old system no expert intervention has been required during data taking.

The control cycle and exception handler synchronization is carried out by OS9 signals rather than by introducing artificial delays, as was the case beforehand. As a consequence the software is much more stable. A recent hardware upgrade and associated software changes were completed in one afternoon rather than several weeks needed for the previous upgrade. This is due to the flexibility inherent in the design. Moving the monitoring software outside the real-time system protects the core control code from frequent intrusion. Monitoring tasks are automated and data driven from configuration files, thus allowing changes by the user without modifying the code. Due to this and the full documentation maintainability is improved.

5. CONCLUSIONS

The OPAL trigger software has been redesigned. The project was completed on schedule and met all requirements, despite the constraints imposed by the timelines and existing interfaces of a running experiment. The successful outcome is a direct consequence of using modern software engineering methodologies.

The customization of the methodologies and tools together with a cookbook like documentation on how to use the SDE makes it easy for new developers to adopt the environment and work with it. The process of this customization carried out by a software engineer and physicists exhibited the difference in their “culture”. Carefully adapting methodologies and tools to the specifics of the project is absolutely necessary. The PSS-05 standards for User Requirements for example are much too detailed for a project like this, nevertheless after customization they proved to be very useful in capturing the user requirements.

The SDE established for the trigger upgrade project now forms the basis for the largest part of the OPAL online software. The redesign of the OPAL event builder made use of it and existing software has been incorporated into the same environment. This did not happen because of a management decision, but because it proved to be useful to the maintainers and developers of the software.

In general the project profited from the methodology and the management techniques, the tools as the instrumentation of the SDE are secondary in that they can be exchanged by similar tools from...
different suppliers. The change in the culture, the “way we usually do things” in the HEP community, is the major point to be achieved to improve software engineering and software quality in HEP experiments.

ACKNOWLEDGMENTS

The successful completion of the project would not have been possible without the active support of the OPAL online coordinators, Per Scharff-Hansen and Frans Meijers, the OPAL trigger coordinators Graham Wilson, and Tara Shears. The help of a professional software engineer, Arash Khodabandeh, with the know how of the CERN/EP/IPT group was absolutely essential for this work, we find it important to have a group acting as internal consultant in CERN to guide us in software engineering matters. In addition we wish to thank Denice Deatrich for writing the new histogram display, and Frans Meijers and Christoph Schwick for their contribution to the trigger monitoring. We gratefully acknowledge the financial support of the Particle Physics and Astronomy Research Council, U.K.

REFERENCES

[5] Macintosh II and AppleTalk are trademarks of Apple Computer Inc., Cupertino, CA 95014, California, USA.
[13] CVSWeb, a utility to browse CVS trees via the web, see http://www.freebsd.org/~fenner/cvsweb/.
[14] Logiscope, available from Verilog SA, 150 rue Nicolas Vauquelin, P.O. Box 1310, F-31106 Toulouse Cedex, France.
Quantum Computing: An Introduction

Tony Hey
Department of Electronics and Computer Science, University of Southampton, Southampton, United Kingdom SO17 1BJ. Email: ajgh@ecs.soton.ac.uk

Abstract
After some remarks on the fundamental physical nature of information, Bennett and Fredkin’s ideas of reversible computation are introduced. This leads on to the suggestions of Benioff and Feynman as to the possibility of a new type of essentially ‘quantum computers’. If we can build such devices, Deutsch showed that ‘quantum parallelism’ leads to new algorithms and new complexity classes. This is dramatically illustrated by Shor’s quantum algorithm for factorization which is polynomial in time in contrast to algorithms for factorization on a classical Turing computer. This discovery has potentially important implications for the security of many modern cryptographic systems. The fundamentals of quantum computing are then introduced - reversible logic gates, qubits and quantum registers. The key quantum property of ‘entanglement’ is described, with due homage to Einstein and Bell. As an illustration of a quantum program, Grover’s database search algorithm is described in some detail. After all this theory, the status of experimental attempts to build a quantum computer is reviewed: it will become evident that we have a long way to go before we can factorize even small numbers. Finally, we end with some thoughts about the process of ‘quantum compilation’ - translating a quantum algorithm into actual physical operations on a quantum system - and some comments on prospects for future progress.

1. Introduction

The fundamental basis of quantum computation is Landauer’s observation that all information is ultimately physical [1, 2]. Information, the 1’s and 0’s of classical computers, must inevitably be recorded by some physical system - be it paper or silicon. Which brings us to the key point. As far as we know today, all matter is composed of atoms - nuclei and electrons - and the interactions and time evolution of atoms are governed by the laws of quantum mechanics. Although the peculiarities of the quantum world may not seem readily apparent at first glance, a closer look reveals that applications of quantum mechanics are all around us (see for example Ref. [3]). As has been emphasized by Minsky [4], the very existence of atoms owes everything not to the chaotic uncertainties of classical mechanics, but rather to the certainties of quantum mechanics with the Pauli exclusion principle and well-defined and stable atomic energy levels! Indeed without our quantum understanding of the solid state and the band theory of metals, insulators and semiconductors, the whole of the semiconductor industry with its transistors and integrated circuits - and hence the computer on which I am writing this lecture - could not have developed. The same can be said about quantum optics and lasers: huge industries - from optical communications to music and video CDs - have their basis in these intrinsically quantum technologies.
At bottom then, everything is quantum mechanical and, like Feynman in his visionary 1959 ‘Plenty of Room at the Bottom’ talk [5], we can certainly envisage storing bits of information on single atoms or electrons. However, these microscopic objects do not obey Newton's Laws of classical mechanics: instead, they evolve and interact according to the Schroedinger equation, the ‘Newton's Law’ of quantum mechanics. In fact, we know now that even this is only a suitable approximation for everyday speeds and energies: at high speeds and energies, we must use the Dirac equation and Einstein's relativity, with its predictions of relativistic mass increase and particle-antiparticle creation, must be taken into account. However, for most of our everyday concerns, it is safe for us to ignore these complications and use the non-relativistic version of quantum mechanics embodied by Schroedinger's equation.

Information is ultimately not an abstract concept - it must be recorded and stored on media that are fundamentally quantum mechanical. We must therefore broaden our definition of information as merely a string of 1's and 0's and examine the consequences of the quantum nature of media for information. The implications of this new field of quantum information theory are still being explored and may yet deliver more surprises. However, to introduce quantum computing, we shall only need a few quantum concepts and principles. But before we turn to a discussion of qubits and the like, we must now make an apparently puzzling diversion and introduce some ideas of Ed Fredkin and Charles Bennett about reversible computing and reversible logic gates.

2. REVERSIBLE COMPUTING

In 1973, Charles Bennett of IBM Research made a remarkable discovery [6]. Classical computation can be broken down into a series of steps, each logically reversible, and this in turn allows physical reversibility of the computation. This result has implications for the energy dissipated by the computation. Rolf Landauer, Bennett's long-term colleague and mentor, had earlier shown that it is the act of discarding information that incurs an unavoidable energy loss. This is Landauer's Principle and, for example, this is now central to our current understanding of the problem of Maxwell's Demon as given by Bennett [7, 8]. Bennett's result means that we can arrange our computer to calculate reversibly, very slowly, with an energy as small as we please. In his lectures on computation in the 1980's [9], Feynman discusses a reversible computer that calculates for a few steps, then drifts back a bit, ‘uncalculating’ as it goes, before it drifts forward again to eventually complete the calculation with almost zero energy loss.

To build such a reversible computer requires us to use new types of logic gates that are reversible, i.e. from the output of the gate one can reconstruct the input. It is easy to see that a conventional AND gate is not reversible. If the output of an AND gate is 0, the signals on the two input wires could be any one of three possibilities - 00, 01 and 10. The possibility of reversible logic gates was considered by Fredkin and Toffoli nearly 20 years ago [10]. Let us consider a simple example. The truth table for a classical NOT gate is shown below (Fig. 1). It is clearly reversible: from its output we can deduce its input. For this reason Feynman prefers to use the symmetrical notation for a NOT gate shown in Fig. 2. Two NOT gates put back to back evidently bring us back to the same place and manifestly demonstrate the reversibility. Consider now the two-input gate shown in Fig.3. This is called a ‘Controlled NOT’ or CN gate, since the NOT operation on the lower input line is only operative when there is a ‘1’ on the upper input: a ‘0’ on the upper input means that the lower bit passes through unchanged. In effect, what appears on the lower output is just the XOR operation on the two input bits (Fig. 4). However, the CN gate is more than just an XOR gate since we retain information about the control bit. This is a general feature of reversible gates: the price for reversibility is that we need to carry round extra bits of information. But, because we are not discarding any information, such a gate is, in principle, more energy efficient than a classical XOR
<table>
<thead>
<tr>
<th>A</th>
<th>NOT A</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 1 Truth Table for NOT gate.

![NOT gate diagram]

Fig. 2 Alternative symbols for NOT gate.

![Controlled NOT or CN gate diagram]

Fig. 3 Controlled NOT or CN gate.

gate. Again, as shown in Fig. 5, the CN gate can be shown to be manifestly reversible by putting two CN gates back to back. Any logical operation can be built from one of several complete sets of classical logic gates - a choice from NOT, AND, OR, XOR, NAND and so on. Similarly, one can show that there are complete sets of reversible gates that allow us to perform any logic operation. In fact, we need more than just the CN gate: we can add a Controlled Controlled NOT (CCN) or ‘Toffoli’ gate (Fig. 6) or a more complicated Fredkin exchange gate (Fig. 7).

Why do we care about all this? Well for one thing it is possible that use of such gates may one day be needed to reduce power consumption of microprocessors implemented in CMOS silicon technology. At present, the Intel Pentium discards something like 100,000 bits per flop with each discarded bit incurring at least the minimum Landauer energy loss [11]. In our case, however, we are interested because the laws of quantum physics are reversible in time. This guarantees that probability is conserved as a state evolves with time. Technically speaking, the Schroedinger time evolution operator is unitary and preserves the norm of quantum mechanical states (see below). To build a quantum computer with quantum states evolving according to the Schroedinger equation therefore necessarily requires us to use realisations of reversible logic gates.
<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a'</th>
<th>b'</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Fig. 4 Truth Table for Controlled NOT gate.

Fig. 5 CN gates are reversible.

Fig. 6 Controlled Controlled NOT, CCN or Toffoli gate.

Fig. 7 Fredkin Exchange gate.
3. QUANTUM COMPLEXITY

Complexity is the study of algorithms. The ‘universality’ of Turing Machines makes it possible for computer scientists to classify algorithms into different ‘complexity classes’. For example, multiplication of two N x N matrices requires an operation count that grows like N^3 with the size of the matrix. This can be analysed in detail for a simple Turing machine implementation of the algorithm. However, the important point about ‘universality’ is that although you may be able to multiply matrices somewhat faster than on a Turing machine, you cannot change from an N^3 growth of operations no matter what Pentium chip or special purpose matrix multiply hardware you choose to use. Thus algorithms, such as matrix multiply, for which execution time and resources grow polynomially with problem size, are said to be ‘tractable’ and in the complexity class ‘P’. Algorithms for which time and resources are found to grow exponentially with problem size are said to be ‘intractable’. There are many subtleties to this classification scheme: the famous ‘Travelling Salesperson Problem’, for example, is in the rather mysterious complexity class ‘NP’. The book by David Harel [12] contains an excellent introduction to this subject.

What has this to do with quantum information and quantum computers? In 1985 David Deutsch pointed out that since a quantum computer was not a Turing machine there was the possibility of new complexity classification of algorithms [13]. As we will see, quantum computers evolve a coherent superposition of quantum states so that each of these states could follow a distinct computational path until a final measurement is made at the output. It is therefore certainly conceptually possible that at least for some problems, quantum computers could surpass the power of classical Turing computers. The first speculation that this might be so is probably due to Feynman in 1981 [14]. However, it was not until 1994 that interest in this subject exploded after Peter Shor's discovery of a new quantum algorithm for factorizing large numbers [15].

Mathematicians believe (although it has yet to be proved) that the number of steps required on a classical computer to factorize a number with N decimal digits grows exponentially with N. Since the computational work required grows very rapidly, the difficulty of factorizing very large numbers has been made the basis of the security of the RSA encryption method (see Ref. [13] for a good review of encryption techniques). This system is widely used to protect electronic bank accounts, for example. The significance of Shor's result was that his algorithm, running on a quantum computer, could solve the factorization problem in polynomial time. What this could mean for the RSA cryptographic system may be illustrated by the time required to factorize a 129 digit number known as RSA129 [16]. In 1994 this required 5000 MIPS-years of computer time to factorize into its 64 and 65 bit prime factors, using over 1000 workstations over a period of 8 months. A quantum computer using Shor's algorithm with a clock speed of 100 MHz could factor RSA129 in a few seconds. This explains the interest of various ‘secret’ government agencies around the world in the feasibility of building quantum computers!

4. QUBITS AND QUANTUM GATES

Instead of using high and low voltages to represent the 1's and 0's of binary data, there is no reason in principle for us not to be able to any two state quantum system. Two commonly discussed possibilities are the two spin states of an electron:

\[ |1\rangle \quad \text{and} \quad |0\rangle \quad \text{as} \quad \uparrow \quad \text{and} \quad \downarrow \]
or two polarization states of a photon:

\[ |1\rangle \quad \text{and} \quad |0\rangle \quad \text{as} \quad H \quad \text{and} \quad V \]

The time evolution of a quantum system is usually well approximated by the Schrödinger equation. In a coordinate space representation, for example, the Schrödinger equation is a linear partial differential equation with the property that any linear superposition of eigenfunctions is also a solution. This superposition property of quantum mechanics means that the general state may be written as a superposition of eigenstates. In the case of our 2-state quantum system the general state may be written as:

\[ |\psi\rangle = \alpha |1\rangle + \beta |0\rangle \]

According to the standard interpretation of quantum mechanics, any measurement (of spin or polarization) made on this state will always yield one of the two eigenvalues with no way of knowing which one. If we prepare an ‘ensemble’ of identical systems then quantum mechanics assures us that we will observe result ‘1’ with probability \( |\alpha|^2 \) and result ‘0’ with probability \( |\beta|^2 \). Normalization of the state to unity guarantees:

\[ |\alpha|^2 + |\beta|^2 = 1 \]

and this normalization and hence the probability interpretation is maintained by any unitary operator \( U \) defined by the property:

\[ U^+ U = 1 \]

Information stored in a 2-state quantum system is called a quantum bit or ‘qubit’: besides storing classical ‘1’ and ‘0’ information there is also the possibility of storing information as a superposition of ‘1’ and ‘0’ states.

We can define quantum analogues of classical reversible gates by means of unitary operators acting on the qubit basis states. For example, a quantum version of the NOT operator may be defined as follows:

\[ U_{\text{NOT}} |1\rangle = |0\rangle \]

\[ U_{\text{NOT}} |0\rangle = -|1\rangle \]

The phase is chosen for consistency of interpretation in terms of rotations of a spin half particle. The NOT gate corresponds to a 180 degree spin rotation. An overall phase makes no
difference to the probability of measuring the particular basis state although any relative phase difference does affect measurements which depend on the interference between the two basis states.

We now see two possible quantum generalisations compared to computation with classical bits. First, we can perform unitary operations on coherent linear combinations of the two basis states:

\[ U_{\text{NOT}} \frac{1}{\sqrt{2}} (|1\rangle + |0\rangle) = \frac{1}{\sqrt{2}} (|1\rangle - |0\rangle) \]

Second, we can consider operations on qubits that have no classical analogue. For example, Deutsch introduces the ‘Square Root of NOT’ operator defined by:

\[ (U_{\text{SRN}})^2 = U_{\text{NOT}} \]

\[ U_{\text{SRN}} |1\rangle = \frac{1}{\sqrt{2}} (|1\rangle + |0\rangle) \]

\[ U_{\text{SRN}} |0\rangle = \frac{1}{\sqrt{2}} (-|1\rangle + |0\rangle) \]

In physical terms, such an operation merely corresponds to a 90 degree spin rotation. Generalizing away from this specific spin interpretation, a transformation that takes a basis state and transforms it into a linear combination of the two basis states is very useful in the construction of quantum algorithms and is called a ‘Hadamard’ transformation.

We have considered a single electron system for storing a single qubit. By considering multiparticle systems we can construct quantum registers. Thus an n-bit register may be written as:

\[ |\psi_n\rangle = |1\rangle \otimes |1\rangle \ldots \otimes |f\rangle \equiv |11\ldots f\rangle \]

---

1 The extra minus signs floating around arise from the fact that spin half systems are double-valued representations of the rotation group. A 360 rotation yields the original state apart from an overall minus sign: a 720 rotation is required to return to where we started.
If we now apply our SRN or Hadamard transformation to this state we now generate a superposition of all $2^n$ states:

$$\psi_n' = U_{SRN} \otimes U_{SRN} \cdots \otimes U_{SRN}|11...1\rangle$$

$$= \frac{1}{2^{n/2}} (|1\rangle + |0\rangle) \otimes (|1\rangle + |0\rangle) \cdots \otimes (|1\rangle + |0\rangle)$$

$$= \frac{1}{2^{n/2}} \{ |11...1\rangle + |11...0\rangle + ... + |00...0\rangle \}$$

In other words, by applying a linear number of operations to the quantum register we are able to generate a register state with an exponential ($2^n$) number of terms. The ability to create such superpositions is one of the key properties that gives quantum parallel processing its power.

We now seem to have all the ingredients - logic gates and registers - to construct a quantum computer. However, neither reversible gates nor superpositions are specifically quantum mechanical. Quantum algorithms derive their remarkable power from one intrinsically quantum phenomenon that we have not so far considered. This is the property called quantum entanglement and, as we shall see, takes us to the very heart of the peculiarities of quantum mechanics.

5. **THE ‘EPR’ PARADOX AND QUANTUM ENTANGLEMENT**

As is well known, Einstein was suspicious of the probabilities inherent in quantum mechanics. In the famous ‘Bohr-Einstein debate’ he tried unsuccessfully to pinpoint an intrinsic contradiction in quantum theory. The climax of this debate was his formulation, with Podolsky and Rosen, of a situation in which one of the essential peculiarities of quantum mechanics was exposed [17]. A modern variant (due to Bohm) of the argument of Einstein, Podolsky and Rosen goes as follows [18]. Imagine we have an elementary particle with zero charge and spin - such as a neutral pion - at rest, which then disintegrates into a spin 1/2 electron and a spin 1/2 positron (Fig. 8). Since angular momentum is conserved in this decay, the two spin half particles must together combine to form a spin zero state. Thus if we measure the electron spin to be up, for example, we know that the positron spin must be down - and vice versa.
So what is the problem? Well, the electron and positron are separating rapidly in opposite directions (conservation of linear momentum). If we make the first spin measurement on the electron, we could in principle measure the positron spin before even a light signal had time to communicate to the positron whether its spin has to be up or down! In fact, no matter what we do, we always find perfect anti-correlation between the spins, even though there could have been no physical communication between the two particles. Einstein thought that this demonstrated that the spins of the two particles are therefore not indeterminate before measurement but are actually ‘elements of physical reality’. According to Bohr’s ‘Copenhagen’ interpretation of quantum mechanics, it is meaningless to talk of the spin direction of the particles until you make a measurement. This is the truly startling point about quantum mechanics: orthodoxy has it that there is no objective reality (a reality independent of an ‘observer’) for the electrons and their spins! Einstein would have none of this and thought that things must really be predetermined in advance of the measurement. In other words, although our present formulation of quantum mechanics has the spins as only having a probabilistic value, and since ‘spooky, faster than light’ signaling is out of the question, there must be some ‘hidden variables’ that make the directions of the spins predetermined from the outset. After several months of frantic activity devising a response to Einstein’s challenge, Bohr declared the EPR paradox not to be a paradox at all and argued essentially that quantum mechanics demands that you are only allowed to treat the electron-positron system as a single quantum system. And there the matter rested, as a rather abstract and philosophical debate about hidden variables and objective reality - since neither side denied that quantum mechanics worked as a predictive framework. Until John Bell entered the debate.

John Bell’s great contribution was to devise a way of putting these two views - hidden variables/objective reality and quantum mechanics/no objective reality - to an experimental test. In our discussion above, we only discussed measuring spins in the ‘up’ and ‘down’ direction. What happens if we measure ‘up/down’ for the electron but ‘left/right’ for the positron. This is easy to calculate according to quantum mechanics. If the electron is found to be ‘up’, the positron state must be ‘down’ for our zero spin initial state, and by standard quantum mechanics a down state may be written as an equal superposition of ‘left’ and ‘right’ eigenstates. Thus a measurement of the ‘right/left’ kind on the positron would yield right or left with equal probabilities. John Bell’s contribution, as he was proud of saying, was to consider the correlations predicted for spin measurements not at right angles but at an angle of 37 degrees, say. In this case, the probabilities for ‘up’ and ‘down’ along this new direction are now not equal and are not purely random. What Bell was able to prove was that the correlations predicted by quantum mechanics are greater than could be

---

2 It is just this property of quantum mechanics that is used as the basis for provably secure key distribution in quantum cryptographic systems[19].
obtained from any local hidden variable theory - where local means there is no faster than light signaling or any other peculiar, acausal behaviour [20]. Unfortunately for Einstein, Alain Aspect and co-workers, in a famous series of experiments, demonstrated (to most people’s satisfaction) that Bell’s hidden variable inequalities are violated and Nature appears to obey quantum mechanics [20].

Why have we made this apparent diversion to discuss the EPR paradox? The reason is that the EPR state of the electron and positron is an example of an ‘entangled state’. If we write the spin zero spin state in terms of the spin states of particles 1 and 2 we have:

$$|S = 0\rangle = \frac{1}{\sqrt{2}} (|\uparrow\rangle_1 |\downarrow\rangle_2 - |\downarrow\rangle_1 |\uparrow\rangle_2)$$

In the EPR case these two particles are rapidly flying apart. The key point about an entangled state is that it is not possible to write such a state as a simple product state of particle 1 and particle 2. Particle 1 is not in a definite spin state - the spin information is shared between the two particles. This is an example of what is sometimes called an ‘entangled qubit’ or just an ‘e-bit’. The important thing to remember is that it is with such states that quantum mechanics shows its bizarre non-local power.

Why are entangled states of relevance to quantum computing? Consider the action of a quantum CN gate:

$$U_{CN}$$

If we apply this transformation to the following product state we generate precisely a state of the EPR form:

$$U_{CN} \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) |0\rangle = \frac{1}{\sqrt{2}} (|0\rangle |0\rangle + |1\rangle |1\rangle)$$

It is the sharing of two halves of an entangled pair that makes possible such things as ‘quantum teleportation’ [21]. In this case, interacting with one half of an EPR pair affects the other half in a non-local way. This remarkable non-local nature of quantum mechanics is also an essential ingredient of quantum algorithms on quantum computers.
6. GROVER’S SEARCH ALGORITHM: AN EXAMPLE OF A QUANTUM ALGORITHM

Consider the problem of searching a database with \( N \) names in a random order for the telephone number of a friend. Classically this would require on average \( N/2 \) steps to find the required entry since there is no smarter way than a brute force \( O(N) \) search. Lov Grover was able to devise a quantum algorithm to search an analogous quantum database with \( N \) items in \( O(\sqrt{N}) \) steps [22]. Although this quantum algorithm does not change the complexity class it still provides significant speed-up for large \( N \). The problem may be formulated as follows. We have a system with \( N = 2^L \) states. Each state can therefore be labeled as an \( L \)-bit string \( S_1, S_2, \ldots, S_N \). The state we want is \( S_m \) which satisfies the condition:

\[
C(S_m) = 1 \text{ for } n = m \\
C(S_n) = 0 \text{ for } n \neq m
\]

The problem is to identify the state \( S_m \).

Grover’s algorithm specifies the following steps:

1. Start with \( L \) qubit register in state

\[
|0\rangle = |00\ldots0\rangle
\]

2. Apply Hadamard transformation to generate a superposition of all possible states

\[
U_H |00\ldots0\rangle = \frac{1}{2^{L/2}} \left( |0\rangle + |1\rangle \right)^L = \frac{1}{2^{L/2}} \sum_{n=0}^{2^L-1} |n\rangle
\]

3. **DO FOR \( \sqrt{N} \) TIMES**

4. Apply the operator \( U_m \) defined by

\[
U_m |n\rangle = |n\rangle \text{ for } n \neq m \\
U_m |n\rangle = -|n\rangle \text{ for } n = m
\]

5. Apply Grover’s ‘Diffusion’ operator

\[
U_D = U_H U_0 U_H
\]

6. **END DO**

7. Measurement yields state \( S_m \) with high probability
Although this algorithm may appear rather cryptic, the effect of these operations is in fact rather simple. The diffusion operator corresponds to a reflection of all the amplitudes about their mean. Because the sign of the amplitude we want has been reversed, this operation amplifies this amplitude at the expense of the others. This is illustrated below for the case $N = 8$, $L = 3$ (Fig. 9).

![Diagram](image-url)

**Fig. 9** Representation of amplitudes and operators for $N=8$ $L=3$ Grover Search.

(Assumed marked bit is $m=3$.)

Boyer et al. [23] have pointed out that Grover's algorithm is one of a general class of ‘amplitude enhancement’ quantum algorithms.

What uses could there be for Grover's algorithm? It will not be very useful for searching a conventional database since it would first be necessary to transfer all the data into a quantum database and this in itself is an $O(N)$ operation. However, it could be effective in cryptoanalysis. In the DES encryption system, the security is ensured by the time required to search a large array of keys. Using Grover's algorithm, a quantum computer could reduce the search time from thousands of years to minutes.
7. EXPERIMENTAL QUANTUM COMPUTING: STATE OF THE ART

This is fast moving field and several different physical realisations of quantum logic gates are being explored by groups in many different countries. The front-runners at the moment are ion traps, cavity QED and NMR ‘ensemble’ quantum computing. The group of David Wineland in Boulder have recently demonstrated a realisation of a CN gate using the two lowest energy levels of two Be ions to realise the two qubit states [24]. The ions are confined in a linear array in an ion trap and cooled to very low temperature using laser cooling techniques. The coupling between the two ions is provided by the vibrational modes of the two ions and Wineland and his group have successfully cooled the system to its vibrational ground state to implement a CN gate. The Caltech group have used an atom interacting with photons in a cavity to demonstrate conditional phase rotations for a two qubit system [25]. Both these approaches attempt to manipulate individual qubits directly. By contrast, an approach using conventional NMR machines manipulates many molecules in bulk. There is no direct control of the qubits of individual molecules but clever manipulation of the NMR operations allows one to effectively isolate pure qubit states. Several groups have investigated quantum systems containing two or three qubits [26-28]. More recently, Jones and co-workers in Oxford have successfully implemented both Grover’s and Deutsch’s algorithms on a two qubit system [29, 30]. Although it seems clear that it will be possible to build and implement quantum algorithms on small numbers of qubits using these technologies, it is by no means clear that any of these will scale to the sort of numbers required for factorizing large numbers.

Most recently Paul Kwiat and co-workers at Los Alamos National Laboratory have implemented Grover’s search algorithm using conventional optical interferometers [31]. In this realisation the two qubits are the two photon polarizations and the two directions through the interferometer. Thus although one can demonstrate quantum gates and algorithms using this technique, no multi-particle entanglement is involved. It is therefore no surprise that increasing the number of qubits using this approach requires exponentially increasing resources.

Several authors have speculated about using solid state devices such as quantum dots or SQUIDs but there seem to be great difficulties in both the control and readout of individual qubits and also in isolating the quantum ‘system’ from the ‘environment’. This last problem is the ‘decoherence’ problem. The coupling to the environment sets bounds to the length of time one can allow the quantum computer to calculate and keep meaningful phase relations between the different states. An exciting futuristic but potentially feasible scheme using carefully positioned phosphorus impurity nuclei in isotopically pure silicon semiconductor has been proposed by Kane [32]. It will be some years, if ever, before such an approach delivers a working quantum computer.

8. CONCLUSIONS

There are many exciting avenues to be explored involving computer scientists, quantum physicists and electronic and photonic engineers. One example has been provided by Butler and Hartel at Southampton [33]. They have shown how Grover’s search algorithm can be expressed in terms of a probabilistic version of Dijkstra’s wp calculus and derived closed forms for its convergence. Another example is the new field of ‘quantum compilers’! Quantum compilation is the business of translating an abstract quantum algorithm down to operations in a given implementation technology. For NMR, for example, a Hadamard transformation must be translated into a specific set of NMR magnetic field pulses.

As we have seen, the extraordinary power of quantum algorithms seems to be derived from the properties of multiparticle entangled states. This is where the peculiar non-local behaviour of
quantum mechanics enters the game. In addition, for a practical quantum computation to survive interactions with the environment and tolerate slightly inaccurate quantum gate operations, the question of error correction must be addressed. Surprisingly, Shor [34] and Steane [35] have independently proposed schemes that show that quantum error correction is indeed possible in principle - something that had hitherto been doubted. Again, entanglement is at the heart of these error correction schemes.

In his 1981 talk [14] in which he first proposed the idea of a quantum computer, Feynman confessed that he was “not sure if there is a real problem with quantum mechanics.” He was also not clear whether quantum computers could be made or would ever do anything useful. But he thought that quantum computation was a wonderful problem to “squeeze the difficulty of quantum mechanics into a smaller and smaller place.” Since quantum computation relies so heavily on the non-local aspect of quantum theory we can extend and stress the theory in new and exciting ways. We may have the foundations of a new multibillion industry or we may find the first clues towards a theory that may eventually supplant quantum mechanics! Both possibilities are exciting.

REFERENCES

Students

J.C. Aparicio, LIP, Lisboa, Portugal
A. Aubord, Observatoire de Geneve, Versoix, Switzerland
M. Christian Beharrell, CERN, Geneva, Switzerland
J. Beringer, Santa Cruz Institute for Particle Physics, Santa Cruz, USA
L. Bertolotto, CERN, Geneva, Switzerland
N. Bisenov, Institute of Nuclear Physics, Ulughbek, Tashkent, Uzbekistan
D. Bocian, Institute of Nuclear Physics, Krakow, Poland
A. Bogdanovich, Budker Institute of Nuclear Physics, Novosibirsk, Russia
C. Bordeanu, National Institute for Physics & Nuclear Engineering, Bucharest-Magurele, Romania
C. Brew, Rutherford Appleton Laboratory, Particle Physics Department, Didcot, UK
L. Brunel, Kossuth University, Debrecen, Hungary
T. Budavari, Eotvos Lorand University, Budapest, Hungary
E. Cecchin, Universita la Sapienza, Roma, Italy
C. Centoli, ENEA - Centro Ricerche Frascati, Frascati (Roma), Italy
D. Chamont, Ecole Polytechnique - LPNHE, Palaiseau, France
S. Coelho da Silva, LIP, Lisboa, Portugal
M. Collados Polidura, CERN, Geneva, Switzerland
J.P. Conceição, LIP, Lisboa, Portugal
J. Conde Gonzalez-Carrascosa, CERN, Geneva, Switzerland
P. Costa Pinto, Universidade Nova de Lisboa, Monte de Caparica, Portugal
P. Crespo, University of Coimbra, Coimbra, Portugal
N. D'Ambrosio, Istituto Nazionale di Fisica Nucleare, Napoli, Italy
G. Della Ricca, Istituto Nazionale di Fisica Nucleare, Trieste, Italy
M. Dressel, Max-Planck-Institit für Physik, Muenchen, Germany
I. Dumanoglu, Cukurova University, Balcali/Adana, Turkey
F. Estrella-Cainglet, University of the Philippines, Diliman, Philippines
J. Flammer, DESY, Hamburg, Germany
F. Garufi, Istituto Nazionale di Fisica Nucleare, Napoli, Italy
F. Gentile, Istituto Nazionale di Fisica Nucleare, Perugia, Italy
A. Go, CERN, Geneva, Switzerland
A. Gomes, LIP, Lisboa, Portugal
J. R. Gonçalo, Universidade de Coimbra, Coimbra, Portugal
I. Gonzalez, Instituto de Fisica de Cantabria, Santander, Spain
E. Grancher, CERN, Geneva, Switzerland
R.L. Groza, Institute for Physics and Nuclear Engineering, Bucharest-Magurele, Romania
M. Haapakorpi, Tampere University of Technology, Tampere, Finland
W. Hoschek, Institute of Applied Computer Science, Linz, Austria
S. Hurling, DESY, Hamburg, Germany
J. Ivarsson, Institute of Physics, Lund, Sweden
J. Iwanicki, Warsaw University, Warszawa, Poland
L. Jahnel, Laboratorio Nacional de Luz Sincrotron - LNLS, Campinas - S. Paulo, Brazil
N. Jardim Nunes, University of Madeira, Funchal, Portugal
A. Kazarov, Petersburg Nuclear Physics Institute (PNPI), Gatchina, Russia
J. Klem, The Helsinki Institute of Physics, Helsinki, Finland
J. Lac, Netherlands Industrial Property Office, Rijswijk, The Netherlands
M. Maggiora, University of Turin, Turin, Italy
M. Mandl, CERN, Geneva, Switzerland
A. Masserot, LAPP, Annecy-le-Vieux, France
C. Matthey, University of California Los Angeles, Los Angeles, USA
J. Carlos Oliveira Morgado, LIP, Lisboa, Portugal
O. Nix, Institut für Hochenergiephysik, Heidelberg, Germany
M. Peryt, Warsaw University of Technology, Warszawa, Poland
K. Petrov, Bogolyubov Institute for Theoretical Physics, Kiev, Ukraine
A. Radu, Institute of Physics and Nuclear Engineering, Bucharest, Romania
N. Ratnikova, ITEP, Moscow, Russia
A. Röllert, Stockholm University, Stockholm, Sweden
H. Saarikoski, University of Helsinki, Helsinki, Finland
M. Schaller, University of Innsbruck, Innsbruck, Austria
S. Scharein, Humbold University of Berlin, Berlin, Germany
O. Schwarzer, DESY, Hamburg, Germany
D. Schweiger, University of Innsbruck, Innsbruck, Austria
M. Sessler, University of Mannheim, Mannheim, Germany
V. Singh, Banaras Hindu University, Varanasi, India
H. Singpiel, University of Mannheim, Mannheim, Germany
C. Von Praun, CERN, Geneva, Switzerland
M. Wittgen, University of Mainz, Mainz, Germany
E. Wolff, University of Hamburg, Hamburg, Germany