1995 CERN SCHOOL OF COMPUTING

Arles, France
20 August – 2 September 1995

PROCEEDINGS

Editor: C.E. Vandoni
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, 1995

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.

ISSN 0304-2898
ISBN 92-9083-076-X
1995 CERN SCHOOL OF COMPUTING

Arles, France
20 August – 2 September 1995

PROCEEDINGS
Editor: C.E. Vandoni
Abstract

These proceedings contain a written account of the majority of the lectures given at the 1995 CERN School of Computing.

The Scientific Programme was articulated on 8 main themes:

- Human Computer Interfaces
- Collaborative Software Engineering
- Information Super Highways
- Trends in Computer Architecture/Industry
- Parallel Architectures (MPP)
- Mathematical Computing
- Data Acquisition Systems
- World-Wide Web for Physics

A number of lectures dealt with general aspects of computing, in particular in the area of Human Computer Interfaces (computer graphics, user interface tools and virtual reality). Applications in HEP of computer graphics (event display) was the subject of two lectures.

The main theme of Mathematical Computing covered Mathematica and the usage of statistics packages.

The important subject of Data Acquisition Systems was covered by lectures on switching techniques and simulation and modelling tools.


Different aspects of Object Oriented Information Engineering Methodology and Object Oriented Programming in HEP were dealt in detail also in connection with data acquisition systems.

On the theme “Trends in Computer Architecture and Industry” lectures were given on: ATM Switching, and FORTRAN90 and High Performance FORTRAN.

Computer Parallel Architectures (MPP) lectures dealt with very large scale open systems, history and future of computer system architecture, message passing paradigm, features of PVM and MPI.
Preface

The eighteenth CERN School of Computing took place in Arles, France, from 20 August to 2 September, 1995, and brought together 21 lecturers and 68 students from 17 different countries and of 21 different nationalities. The School was organised in collaboration with the Institut National de Physique Nucléaire et de Physique des Particules (IN2P3) which is part of the Centre National de Recherche Scientifique, France. The organisers would like to thank the members of the local organising committee Dr. J. Ganouna, IN2P3, Villeurbanne, Dr. G. Grosdidier, LAL, Orsay, and especially Dr. F. Etienne, of the Centre de Physique des Particules (CPPM), Marseille as without his very active help and efficient organisation, the 1995 School would not have come into being.

The School was honoured by the presence of Prof. H. Curien, President of the CERN Council, and by that of Mr. C. Détraz, Director of the Institut National de Physique Nucléaire et de Physique des Particules (IN2P3), Paris, who set the tone for the School by their interesting and informative speeches at the Opening Ceremony. Professor A. Hey of Southampton University, Chairman of the Advisory Committee to the CERN Schools of Computing, is also to be thanked for his overview of the aims of the School.

We would also like to thank the following people for their help and their warm welcome: Mr. M. Vauzelle, Mayor of Arles; Mr. J.-P. Camoin, former Mayor of Arles; Mr. H. Schiavetti, First Deputy Mayor; Mr. L. Sayn Urpar, former First Deputy Mayor; Mr. H. Dworak, Deputy Mayor for Sport; Mrs. D. Pralong and Mr. L. Guedj (External Relations, Arles); Mr. Calibert (Director, University Centre Espace Van Gogh, Arles) and his staff; Mr. Martre (and his son Florian) who helped out with the Cyber café which was open to the public, the town of Arles. We are particularly grateful to the Municipality for allowing us to freely use the Espace Van Gogh (formerly the Hotel Dieu) for the 1995 CERN School of Computing, as well as the municipal sports facilities.

The lecture programme (46 one hour lectures, supplemented by tutorials) was varied and was greatly appreciated by the students. All lecturers are sincerely thanked for their efforts in preparing and delivering their lectures, for providing written accounts of their presentations, and for their availability for discussion with the students. The lecture programme was complemented by very demanding practical work. Most of the students tackled the practical exercises with great interest and enthusiasm. Drs. R. Barlow, J. Gallop, J.-M. Le Goff, R. Jones, M. Trott and P. Palazzi, together with his colleagues M. Dönzselmann, A. Khodabandeh, P. Pinasseau and B. Rousseau, are all greatly thanked for the very hard work that they put into preparing their practical exercises and tutorials.

This year, three evening lectures, which were open to the public, were organised and we would like to thank Mr. R. Cailliau (CERN, Geneva), Dr. A. Tilquin (CPPM, Marseille) and Mr. E. Todd (INED, Paris) for having accepted this charge.

Our thanks also go to Mr. G.V. Frigo, CERN, for designing the layout of the poster and the School booklet.

Mr. M. Ruggier, CERN, is thanked for setting up the CSC home page on the WorldWideWeb.
Mr. C. Isnard, CERN and Mr. J. Bernier, CPPM Marseille were of great help to us in setting up the MBONE connection and our grateful thanks go to them.

Messrs. R. Bazzoli, C. Meessen and M. Ricard from CPPM, Marseille, together with Messrs. P. Challeat, G. Drevon, D. Charney and G. Farrache from IN2P3, Villeurbanne, are to be congratulated on their very hard and efficient work in installing workstations and the network.

The School is indebted to the following companies for the loan of equipment and software:

For the loan of workstations: Digital Equipment (Geneva), ESC, Hewlett Packard (France), Hewlett Packard (Switzerland), MIC, Silicon Graphics, Sun, Tektronix.

For the loan of software packages: Ecrin, NAG, RITME

In particular we would like to thank Hewlett Packard (Switzerland), in the person of Mr. M. Bénard, for their sponsorship.

The Scientific Publishers, Addison-Wesley, and the Eurographics Association very kindly provided books for the School and these were incorporated into the Library, together with books which were lent by the CERN Computer Science Library, the Programming Techniques Group, ECP Division and the Users Consultancy Office, CN Division. The Library was much appreciated by the Students.

Accommodation was provided by the Hotel Atrium. The hotel offered excellent service and we wish to thank the Director, Mr. F. Sanchez, for the kindness and efficiency of his personnel.

Three excursions were organised. The first was an afternoon excursion to Les Baux, the Cathedrale d’Images and Glanum; the second was an all day visit to the Tour de Valat (station biologique) situated in the Camargue area - our grateful appreciation goes to Mr. J.P. Taris, Director and to his team for their interesting presentations and warm hospitality. The third (an afternoon visit) was to the Pont de Gard and to Nîmes and Mr. P. Scheurer (General Secretary - Structural Equipment) of the Nîmes municipality is to be congratulated on his informative talk about the Nîmes arena. A short visit to the Institut de Recherche de la Provence Antique (IRPA) was also arranged and our thanks go to the curator, Mr. C. Sintés, for having been our very special guide. These breaks in the programme were very much appreciated.

We express our gratitude to our secretary, Jacqueline Turner, not only for the efforts made during the last twelve months in preparing the School and the hard work done during the School, but also for her invaluable help in preparing these Proceedings.

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

C.E. Vandoni, Editor
Advisory Committee

G. Barreira, Laboratório de Instrumentação e Física Esperimental de Partículas (LIP), Lisbon, Portugal

W. Carena, CERN, Geneva, Switzerland

F. Etienne, Centre de Physique des Particules de Marseille (CPPM), Marseille, France

F. Flückiger, CERN, Geneva, Switzerland

J. Ganouna, IN2P3, Villeurbanne, France

G. Grosdidier, Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France

L.O. Hertzberger, Universiteit van Amsterdam, Amsterdam, The Netherlands

A.J.G. Hey, University of Southampton, Southampton, U.K.

G. Kellner, CERN, Geneva, Switzerland

M. Metcalf, CERN, Geneva, Switzerland

P. Palazzi, CERN, Geneva, Switzerland

J. Turner, CERN, Geneva, Switzerland

C. Vandoni, CERN, Geneva, Switzerland

D.O. Williams, CERN, Geneva, Switzerland
Lecturers

R. Barlow, University of Manchester, Manchester, UK
W. Bauerfeld, DeTeBerkom, Berlin, Germany
S. de Gennaro, CERN, Geneva, Switzerland
D.C. De Roure, University of Southampton, Southampton, UK
M. Dönszelmann, CERN, Geneva, Switzerland
H. Drevermann, CERN, Geneva, Switzerland
T. Engbersen, IBM Zürich Research Lab., Rüschlikon, Switzerland
J. Gallop, Rutherford Appleton Laboratory, Didcot, UK
R. Groves, IBM Corporation, Austin, Texas, USA
M. Haney, University of Illinois, Urbana, USA
R. Jones, CERN, Geneva, Switzerland
J.-M. Le Goff, CERN, Geneva, Switzerland
M. Letheren, CERN, Geneva, Switzerland
H. Lie, INRIA-W3, Sophia Antipolis, France
C. Maillot, Thomson-CSF, Orsay, France
M. Metcalf, CERN, Geneva, Switzerland
D. Quarrie, Lawrence Berkeley Laboratory, Berkeley, USA
B. Rousseau, CERN, Geneva, Switzerland
RD Schaffer, Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France
M. Trott, Wolfram Inc., Champaign, USA
D. Walker, Oak Ridge Laboratory, Oak Ridge, USA
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>iii</td>
</tr>
<tr>
<td>Photography</td>
<td>iv</td>
</tr>
<tr>
<td>Preface</td>
<td>v</td>
</tr>
<tr>
<td>Advisory Committee</td>
<td>vii</td>
</tr>
<tr>
<td>Lecturers</td>
<td>viii</td>
</tr>
<tr>
<td>Contents</td>
<td>ix</td>
</tr>
</tbody>
</table>

**Human Computer Interfaces**

Computer Graphics and Human Computer Interfaces  
*J. Gallop* ......................................................... 1

Virtual Prototyping at CERN  
*S. de Gennaro* .................................................. 13

Event Display: Can We See What We Want to See?  
*H. Drevermann, D. Kuhn, B.S. Nilsson* ...................... 17

**Collaborative Software Engineering**

The RD13 DAQ System and the Object Management Workbench  
*B. Jones* ......................................................... 65

CICERO RD38: A Distributed Information System for HEP Controls  
*J.-M. Le Goff* .................................................. 85

Object Oriented Programming and High Energy Physics  
*D. Quarrie* ....................................................... 101

Applying an Object Oriented Approach to Off-line Reconstruction at the LHC  
*A. Schaffer* ..................................................... 117

**Information Super Highways**

First Driving Classes on the Information Superhighway  
*W. Bauerfeld* ...................................................... 131

Information Highway Applications  
*D. De Roure* ...................................................... 139

**Trends in Computer Architecture/Industry**

Brief History of Computer Architecture Evolution and Future Trends  
*R. Groves* ........................................................ 147
High-Speed Switching
   T. Engbersen ................................................................. 161

Parallel Architectures (MPP)

An Introduction to Message Passing Paradigms
   D. Walker ................................................................. 165

Fortran 90 - A Thumbnail Sketch
   M. Metcalf ............................................................... 185

High Performance Fortran
   M. Metcalf ............................................................... 193

Mathematical Computing

Using a Statistics Package
   R. Barlow ............................................................... 199

Three Lectures on Mathematica
   M. Trott ................................................................. 211

Data Acquisition Systems

Simulation and Modeling Tools in Data Acquisition System Design for
Future High Energy Physics Experiments
   M. Haney ............................................................... 237

Switching Techniques in Data Acquisition Systems for Future Experiments
   M. Letheren ............................................................. 245

WorldWideWeb for Physics

World-WideWeb Technology
   H. Lie ................................................................. 277

Publishing on the Web
   B. Rousseau .......................................................... 279

Interfacing to the Web
   M. Dönszelmann ...................................................... 295

   Students ................................................................. 307
1995 CERN SCHOOL OF COMPUTING

Arles, France
20 August – 2 September 1995

PROCEEDINGS
Editor: C.E. Vandoni

GENÈVE 1995
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, 1995

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.

ISSN 0304-2898
ISBN 92-9083-076-X
Abstract

These proceedings contain a written account of the majority of the lectures given at the 1995 CERN School of Computing.

The Scientific Programme was articulated on 8 main themes:

Human Computer Interfaces
Collaborative Software Engineering
Information Super Highways
Trends in Computer Architecture/Industry
Parallel Architectures (MPP)
Mathematical Computing
Data Acquisition Systems
World-Wide Web for Physics

A number of lectures dealt with general aspects of computing, in particular in the area of Human Computer Interfaces (computer graphics, user interface tools and virtual reality). Applications in HEP of computer graphics (event display) was the subject of two lectures.

The main theme of Mathematical Computing covered Mathematica and the usage of statistics packages.

The important subject of Data Acquisition Systems was covered by lectures on switching techniques and simulation and modelling tools.


Different aspects of Object Oriented Information Engineering Methodology and Object Oriented Programming in HEP were dealt in detail also in connection with data acquisition systems.

On the theme “Trends in Computer Architecture and Industry” lectures were given on: ATM Switching, and FORTRAN90 and High Performance FORTRAN

Computer Parallel Architectures (MPP) lectures dealt with very large scale open systems, history and future of computer system architecture, message passing paradigm, features of PVM and MPI.
Preface

The eighteenth CERN School of Computing took place in Arles, France, from 20 August to 2 September, 1995, and brought together 21 lecturers and 68 students from 17 different countries and of 21 different nationalities. The School was organised in collaboration with the Institut National de Physique Nucléaire et de Physique des Particules (IN2P3) which is part of the Centre National de Recherche Scientifique, France. The organisers would like to thank the members of the local organising committee Dr. J. Ganouna, IN2P3, Villeurbanne, Dr. G. Grosdidier, LAL, Orsay, and especially Dr. F. Etienne, of the Centre de Physique des Particules (CPPM), Marseille as without his very active help and efficient organisation, the 1995 School would not have come into being.

The School was honoured by the presence of Prof. H. Curien, President of the CERN Council, and by that of Mr. C. Détraz, Director of the Institut National de Physique Nucléaire et de Physique des Particules (IN2P3), Paris, who set the tone for the School by their interesting and informative speeches at the Opening Ceremony. Professor A. Hey of Southampton University, Chairman of the Advisory Committee to the CERN Schools of Computing, is also to be thanked for his overview of the aims of the School.

We would also like to thank the following people for their help and their warm welcome: Mr. M. Vauzelle, Mayor of Arles; Mr. J.-P. Camoin, former Mayor of Arles; Mr. H. Schiavetti, First Deputy Mayor; Mr. L. Sayn Urpar, former First Deputy Mayor; Mr. H. Dworak, Deputy Mayor for Sport; Mrs. D. Pralong and Mr. L. Guedj (External Relations, Arles); Mr. Calibert (Director, University Centre Espace Van Gogh, Arles) and his staff; Mr. Martre (and his son Florian) who helped out with the Cyber café which was open to the public; the town of Arles. We are particularly grateful to the Municipality for allowing us to freely use the Espace Van Gogh (formerly the Hotel Dieu) for the 1995 CERN School of Computing, as well as the municipal sports facilities.

The lecture programme (46 one hour lectures, supplemented by tutorials) was varied and was greatly appreciated by the students. All lecturers are sincerely thanked for their efforts in preparing and delivering their lectures, for providing written accounts of their presentations, and for their availability for discussion with the students. The lecture programme was complemented by very demanding practical work. Most of the students tackled the practical exercises with great interest and enthusiasm. Drs. R. Barlow, J. Gallop, J.-M. Le Goff, R. Jones, M. Trott and P. Palazzi, together with his colleagues M. Dönszelmann, A. Khodabandeh, P. Pinasseau and B. Rousseau, are all greatly thanked for the very hard work that they put into preparing their practical exercises and tutorials.

This year, three evening lectures, which were open to the public, were organised and we would like to thank Mr. R. Cailliau (CERN, Geneva), Dr. A. Tilquin (CPPM, Marseille) and Mr. E. Todd (INED, Paris) for having accepted this charge.

Our thanks also go to Mr. G.V. Frigo, CERN, for designing the layout of the poster and the School booklet.

Mr. M. Ruggier, CERN, is thanked for setting up the CSC home page on the WorldWideWeb.
Mr. C. Isnard, CERN and Mr. J. Bernier, CPPM Marseille were of great help to us in setting up the MBONE connection and our grateful thanks go to them.

Messrs. R. Bazzoli, C. Meessen and M. Ricard from CPPM, Marseille, together with Messrs. P. Challeat, G. Drevon, D. Charney and G. Farrache from IN2P3, Villeurbanne, are to be congratulated on their very hard and efficient work in installing workstations and the network.

The School is indebted to the following companies for the loan of equipment and software:

For the loan of workstations: Digital Equipment (Geneva), ESC, Hewlett Packard (France), Hewlett Packard (Switzerland), MIC, Silicon Graphics, Sun, Tektronix.

For the loan of software packages: Ecrin, NAG, RITME

In particular we would like to thank Hewlett Packard (Switzerland), in the person of Mr. M. Bénard, for their sponsorship.

The Scientific Publishers, Addison-Wesley, and the Eurographics Association very kindly provided books for the School and these were incorporated into the Library, together with books which were lent by the CERN Computer Science Library, the Programming Techniques Group, ECP Division and the Users Consultancy Office, CN Division. The Library was much appreciated by the Students.

Accommodation was provided by the Hotel Atrium. The hotel offered excellent service and we wish to thank the Director, Mr. F. Sanchez, for the kindness and efficiency of his personnel.

Three excursions were organised. The first was an afternoon excursion to Les Baux, the Cathedrale d’Images and Glanum; the second was an all day visit to the Tour de Valat (station biologique) situated in the Camargue area - our grateful appreciation goes to Mr. J.P. Taris, Director and to his team for their interesting presentations and warm hospitality. The third (an afternoon visit) was to the Pont de Gard and to Nîmes and Mr. P. Scheurer (General Secretary - Structural Equipment) of the Nîmes municipality is to be congratulated on his informative talk about the Nîmes arena. A short visit to the Institut de Recherche de la Provence Antique (IRPA) was also arranged and our thanks go to the curator, Mr. C. Sintés, for having been our very special guide. These breaks in the programme were very much appreciated.

We express our gratitude to our secretary, Jacqueline Turner, not only for the efforts made during the last twelve months in preparing the School and the hard work done during the School, but also for her invaluable help in preparing these Proceedings.

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

C.E. Vandoni, Editor
Advisory Committee

G. Barreira, Laboratório de Instrumentação e Física Experimental de Partículas (LIP), Lisbon, Portugal

W. Carena, CERN, Geneva, Switzerland

F. Etienne, Centre de Physique des Particules de Marseille (CPPM), Marseille, France

F. Flückiger, CERN, Geneva, Switzerland

J. Ganouna, IN2P3, Villeurbanne, France

G. Grosdidier, Laboratoire de l'Accélérateur Linéaire (LAL), Orsay, France

L.O. Hertzberger, Universiteit van Amsterdam, Amsterdam, The Netherlands

A.J.G. Hey, University of Southampton, Southampton, U.K.

G. Kellner, CERN, Geneva, Switzerland

M. Metcalf, CERN, Geneva, Switzerland

P. Palazzi, CERN, Geneva, Switzerland

J. Turner, CERN, Geneva, Switzerland

C. Vandoni, CERN, Geneva, Switzerland

D.O. Williams, CERN, Geneva, Switzerland
Lecturers

R. Barlow, University of Manchester, Manchester, UK

W. Bauerfeld, DeTeBerkom, Berlin, Germany

S. de Gennaro, CERN, Geneva, Switzerland

D.C. De Roure, University of Southampton, Southampton, UK

M. Dönszelmann, CERN, Geneva, Switzerland

H. Drevermann, CERN, Geneva, Switzerland

T. Engbersen, IBM Zürich Research Lab., Rüschlikon, Switzerland

J. Gallop, Rutherford Appleton Laboratory, Didcot, UK

R. Groves, IBM Corporation, Austin, Texas, USA

M. Haney, University of Illinois, Urbana, USA

R. Jones, CERN, Geneva, Switzerland

J.-M. Le Goff, CERN, Geneva, Switzerland

M. Letheren, CERN, Geneva, Switzerland

H. Lie, INRIA-W3, Sophia Antipolis, France

C. Maillot, Thomson-CSF, Orsay, France

M. Metcalf, CERN, Geneva, Switzerland

D. Quarrie, Lawrence Berkeley Laboratory, Berkeley, USA

B. Rousseau, CERN, Geneva, Switzerland

RD Schaffer, Laboratoire de l’Accélérateur Linéaire (LAL), Orsay, France

M. Trott, Wolfram Inc., Champaign, USA

D. Walker, Oak Ridge Laboratory, Oak Ridge, USA
# Contents

Abstract ......................................................................................................................... iii
Photography .................................................................................................................... iv
Preface .............................................................................................................................. v
Advisory Committee ........................................................................................................ vii
Lecturers ........................................................................................................................... viii
Contents ......................................................................................................................... ix

**Human Computer Interfaces**

Computer Graphics and Human Computer Interfaces  
  *J. Gallop* ....................................................................................................................... 1

Virtual Prototyping at CERN  
  *S. de Gennaro* ............................................................................................................ 13

Event Display: Can We See What We Want to See?  
  *H. Drevermann, D. Kuhn, B.S. Nilsson* ...................................................................... 17

**Collaborative Software Engineering**

The RD13 DAQ System and the Object Management Workbench  
  *B. Jones* ...................................................................................................................... 65

CICERO RD38: A Distributed Information System for HEP Controls  
  *J.-M. Le Goff* ............................................................................................................. 85

Object Oriented Programming and High Energy Physics  
  *D. Quarrie* ................................................................................................................... 101

Applying an Object Oriented Approach to Off-line Reconstruction at the LHC  
  *A. Schaffer* ................................................................................................................. 117

**Information Super Highways**

First Driving Classes on the Information Superhighway  
  *W. Bauerfeld* ............................................................................................................... 131

Information Highway Applications  
  *D. De Roure* .............................................................................................................. 139

**Trends in Computer Architecture/Industry**

Brief History of Computer Architecture Evolution and Future Trends  
  *R. Groves* .................................................................................................................... 147
High-Speed Switching
T. Engbersen ........................................................................................................ 161

Parallel Architectures (MPP)

An Introduction to Message Passing Paradigms
D. Walker ............................................................................................................ 165

Fortran 90 - A Thumbnail Sketch
M. Metcalf ........................................................................................................ 185

High Performance Fortran
M. Metcalf ........................................................................................................ 193

Mathematical Computing

Using a Statistics Package
R. Barlow .......................................................................................................... 199

Three Lectures on Mathematica
M. Trott ............................................................................................................. 211

Data Acquisition Systems

Simulation and Modeling Tools in Data Acquisition System Design for Future High Energy Physics Experiments
M. Haney ........................................................................................................... 237

Switching Techniques in Data Acquisition Systems for Future Experiments
M. Letheren ........................................................................................................ 245

WorldWideWeb for Physics

World-WideWeb Technology
H. Lie .................................................................................................................. 277

Publishing on the Web
B. Rousseau ...................................................................................................... 279

Interfacing to the Web
M. Dönszelmann ............................................................................................... 295

Students ............................................................................................................. 307
Computer Graphics and Human Computer Interfaces

J.R. Gallop

*CCLRC Rutherford Appleton Laboratory, Didcot, Oxfordshire, UK*

Abstract

This paper provides a brief summary of some of the topics being covered in the lectures on computer graphics at the CERN School of Computing.

**Keywords:** Computer Graphics.

1. Uses of computer graphics

Computer graphics is now in wide use across a multitude of applications which include science, engineering, education, entertainment and medicine. It can be used:

- as part of the design process: a computer model of a proposed artefact - whether a cassette recording head or a civil airliner - is constructed, assessed, updated and used as the master for consultation, manufacture, maintenance and upgrading.
- to visualize complex data from experimental sources or from a computer simulation.
- to provide information
- to entertain with fast moving and/or complex games

2. A pixellated world

When talking about computer graphics, one commonly sees reference to addressability, RAM, bitmaps, pixels, limited colour maps and so on. These are important, but they represent the limitations inherent in the hardware we have at our disposal.

It is important not to be over-influenced by the discrete limitations of the actual devices. The user wishes to perceive a representation of an idealised model - whether this be an engine part of a civil airliner, a car in a racing game or earth sensing data. The user also wishes to interact with the idealised model - control the view of it, alter it, evaluate it. The task of computer graphics is to allow (indeed encourage) the user’s active engagement with the idealised model and to reduce the inhibitions imposed by the limitations of actual devices. These limitations can include:

- spatial resolution,
- number of colours and brightness levels,
- time.
Most devices in use today are raster devices, which rely on being able to store a pixel value at each cell of a raster. The number of available pixels in each of x and y limits the spatial resolution of the device. The maximum value at each pixel restricts the brightness and range of colour that is possible. We show in this table some typical values for commonly found devices.

<table>
<thead>
<tr>
<th></th>
<th>x * y</th>
<th>dpi</th>
<th>possible pixel values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laser printer</td>
<td>600</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Dye sublimation printer</td>
<td>300</td>
<td>2&lt;sup&gt;24&lt;/sup&gt;</td>
<td></td>
</tr>
<tr>
<td>Publication printer</td>
<td>2400</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Quality display</td>
<td>2000*1500</td>
<td>150</td>
<td>2&lt;sup&gt;24&lt;/sup&gt;</td>
</tr>
<tr>
<td>VR headset</td>
<td>640 * 480</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1: Table of device capabilities**

How many brightness values are enough? Although the human visual system can detect enormous brightness range, (approximately a factor of $10^{10}$ to 1 between bright sunlight and the dimmest detectable list source [1]), it does not do so all at the same time, as the aperture of the eye compensates. The higher end of the dynamic range of a CRT is approximately 200 and the typical range for black and white printing on coated paper is about 100 (black and white on newsprint is 10).

Supposing that the brightness levels are evenly distributed over the dynamic range of the display device, one could make a calculation of the number of levels required (a more detailed discussion appears in [2]). We must realize that the eye perceives the ratio between two intensity levels not the arithmetic difference: thus the perceived difference between 0.1 and 0.2 appears to be the same as between 0.4 and 0.8. The reproduction on the graphics output medium is perceived to be continuous when the ratio is about 1.01. Thus if the dynamic range is d and the number of levels is n, we have

$$1.01^n = d$$
Typical values of $n$ of a good CRT under good conditions are 530 and for black and white on coated paper, 465. But evenly distributing the brightness levels over the dynamic range is difficult to achieve ....

Note that we have been discussing the requirements for continuous tone. If we are using colours to present distinct meanings to the observer, the situation is different. Consider as an example the map of the underground train system of London or Paris. Beyond about 10 colours, distinguishing certain pairs of underground lines is quite difficult.

The limitations of time are partly those of speed - can the processor achieve the task fast enough. However even if the processor is fast, we also have operating system software problems which may prevent us achieving synchronisation and time accuracy. For some applications (video-conferencing, movie playback), synchronisation between vision and sound is vital. For these and other applications (visualizing a time-varying sequence), time accuracy is essential.

Let us recap. The application programmer has an idealised description of the graphics to be presented and the interactions to be allowed. The task of computer graphics software is to overcome the quantising effects and other limitations of actual devices so that the user can interact with the idealised model as faithfully as possible.

3. **2D graphics**

Although 3D graphics is becoming more widely used as hardware capable of supporting it well becomes cheaper, 2D graphics continues to be important for images, schematics, charts and diagrams.

3.1 **The idealised form**

As we discussed, in most applications there is an idealised model of the geometry under consideration. Somehow this needs to be converted to the actual, discrete graphics device.

Some 2D graphics software allows the application program to specify this ideal form and is therefore responsible for achieving the best possible results on the graphics device. Points and vectors are specified in a coordinate system (usually Cartesian) chosen by the application programmer. Colours are specified as real numbers. An example of this software is the ISO standard for 2D graphics programming, of which the most recent version is GKS-94 [3].

Other (lower level) software allows the application program to have close control over the specific pixels of the graphics device. Xlib (in the X Window System) allows this. It is the responsibility of the application program to map from the application's idealised model to the specific points and colours of the graphics device.

3.2 **Output functions**

To allow the application to specify the 2D output in a idealised form, what output functions do we need. In principle, we need a 0D function (draw points), a 1D
function (draw lines) and a 2D function (draw areas). In practice this is too spartan and an application programming interface needs more support than this.

It is instructive to summarise the output functions supplied in GKS-94 as these will be regarded as a minimum for other packages in future.

- **SET OF POLYLINE**: set of curves, each of which is a sequence of connected lines defined by a point sequence
- **SET OF NURB**: set of curves, each of which is a NURB (discussed later)
- **SET OF CONIC SECTION**: set of curves each of which is a conic section
- **POLYMARKER**: set of symbols of one type centred at given positions
- **SET OF FILL AREA**: set of areas each of which is defined by a closed sequence of connected lines
- elliptic arc primitives
- **TEXT**
- **CELL ARRAY**: array of cells with individual colours
- **DESIGN**: a tiling may be extruded through a stencil which may consist of area boundaries, closed contours and others.

### 3.3 2D transformations

Having defined 2D objects by means of some output functions, we need to think about how 2D graphics are transformed. There are two main reasons:

- We may be composing a complex schematic such as a floor plan. It is convenient to define an object shape and then to place it in the schematic. Placing it may involve rotating it and scaling it as well as translating the object to its desired place.

- Having defined the complete floor plan, we may wish to view different parts of it at different magnitudes. We can think of a camera panning, zooming and rotating. In 2D this panning and zooming usually achieved when the coordinates are converted from the 2D world of the application to those of the device. However in 3D, viewing becomes a more complex issue as the camera movements can be more complex.

These 2D transformations can be achieved by the application of straightforward 2D matrix operations. We can represent a point by a vector \([x, y]\) - many introductory texts in fact use the convention of a column vector.

To rotate a point about the origin, multiply the point by a rotation matrix.

To scale a point with respect to the origin, multiply the point by a scaling matrix.

But, to translate a point, add a translation vector. This is unfortunate because (for efficiency and simplicity) we would like to transform the point in one operation. If we could only use matrix multiplication for everything, we could win. What can we do about translation?
The answer is to introduce homogeneous coordinates. We can represent a 2D point by a vector
\[
\begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\]

By taking care with zero, we can establish the convention that
\[
\begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\quad \text{and} \quad
\begin{bmatrix}
    wx \\
    wy \\
    w
\end{bmatrix}
\]
are the same point.

We can make scale and rotate work as before by extending them with 0's and 1's:
\[
\begin{bmatrix}
    x' \\
    y' \\
    1
\end{bmatrix} =
\begin{bmatrix}
    s & t & 0 \\
    u & v & 0 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\]

We can now create a matrix for translation which allows us to use multiplication:
\[
\begin{bmatrix}
    x + tx \\
    y + ty \\
    1
\end{bmatrix} =
\begin{bmatrix}
    1 & 0 & tx \\
    0 & 1 & ty \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\]

This is a greatly simplified view and more detail can be found in the introductory texts [2] and [4].

3.4 Object structure

In many applications it becomes important to group objects so that they can be manipulated together or separately. A group of objects may need to be identified by the user using some pick mechanism and hence selectively made invisible or highlighted in some way. This becomes even more important when we think about 3D.

Various methods have been used. PHIGS (and PHIGS PLUS [5]) uses a method of hierarchical structures. This design is useful for manipulating scenes made up of objects which are made up of a collection of rigid parts.

GKS-94 uses an approach called Name Sets. Each output primitive created through GKS-94 has a set of names associated with it and filters can subsequently be applied. Instead of the hierarchical approach of PHIGS, this can be thought of as a set of layers which can be handled independently. This approach has promise in the field of cartography.
Open GL also uses a hierarchical scene database.

3.5 2D computer graphics file transfer and archival

At some point it becomes necessary to transfer pictures to another system or to archive them. One way is to create an image and archive and store that. There are many image formats and most readers will be aware of them.

Less commonly known about is the Computer Graphics Metafile (CGM). This is an ISO standard which was revised in 1992 [6]. It contains graphics primitives, not just images. Converters and interpreters are available on PCs, Macintosh’s and Unix workstations and a variety of graphics software can create as a CGM as an output driver.

CGM can also fill a hole in the World Wide Web. Images have the problem of not being scalable and also require much network bandwidth even when compressed. Many pictures are better stored in the form of geometric primitives which CGM does.

At present CGM is 2D, but there are moves to produce a 3D version.

4. 3D graphics

3D graphics is now in widespread use. It is in extensive use for Computer Aided Design. Data visualization software often results in an abstract 3D model. Virtual Reality presents a 3D world to the participant and tries to give the best possible sense of “being there”. Most home computer have some 3D graphics software.

Although it is usually not necessary to understand the underlying algorithms, people who are creating application software using 3D graphics need some understanding of the many options. Even people who are using pre-written applications software need some grasp to enable better choices to be made. In this section we introduce some of the problems that need to be thought about.

4.1 Models and views

When discussing 2D graphics, we introduced the idea of composing the floor plan and as a separate operation viewing it through a camera. This distinction between the two processes, which is often blurred in 2D systems, is even more important in 3D.

A 3D graphics system allows scenes to be composed of objects. The objects themselves may be defined in a hierarchical way i.e. there are subobjects. A table may consist of a table top, a number of legs and, if the detail matters to the application, the fixings that hold it together. The scene may be a room which may consist of the walls doors and also the contents of the room. At the lowest level each object needs to be defined in terms of some output primitives.

A 3D graphics system also allows scenes to be viewed and one method is to simulate a camera, with a position and a view direction.
4.2 Output primitives

Output primitives fall into a number of categories (PHIGS PLUS is used here as a source of examples).

Some output primitives are inherited from a 2D system but are available in 3D space. Examples of this are polyline, polymarker and fill area.

Other primitives consist of multiple polygons. Examples are triangular set, triangular set and quadrilateral mesh. An object may be represented entirely in terms of triangles in which case it may be represented by the single primitive triangular set, which allows the 3D graphics system to handle shading and hidden surface removal correctly.

The other category is represented by curves and surfaces. Often the ideal for the application is a curved line or surface. Complex meshes often result from an object that has already been approximated.

There are many ways of representing a curve. If at all complex, it has become common to use a piecewise representation, which avoids a polynomial representation of a high degree.

The B-spline concept is one example of this. It can be uniform or non-uniform, which refers to the spacing of the knots - the knots divide the subinterval over which each piece is defined. A B-spline may consist of non-rational or rational polynomials. With the rational form, weights are applied to control parts of the spline more tightly. The most general of these is the non-uniform, rational B-spline i.e. the NURB. This is found in PHIGS PLUS and other systems.

B-splines have a number of advantages:

- Compared with some other types of curve, it is possible to exert local control of the shape of the curve. This is important in many design applications.
- A NURB can be used to model all conics exactly with a small number of control points.
- NURBS are invariant under perspective transformation.

4.3 3D transformations

In 2D graphics, the operations of translate, scale and rotate are achieved using 3 x 3 matrices and, with the modification we described for translate, using matrix multiplication. This allows a user to expect a graphics system to compose a non-trivial transformation without loss of performance.

The same operations can be used in 3D using 4 x 4 matrices and non-trivial transformations may be composed in the same way.

However in 3D we also have to deal with projecting the 3D scene onto a 2D graphics device. We have to assume a viewer or a camera, pointing at a screen through which the scene can be viewed.
Since we have 3D rotations and translations, we can always transform the 3D coordinate system so that the coordinates are simplified as shown. We seek the (x,y) coordinates of the projection of the object on the screen.

For a perspective projection, this is a straightforward division:

\[(Xv/Z, Yv/Z)\]

and we retain Z for any calculations relying on depth.

For a parallel projection, it is even simpler as there is no division.

4.4 Rendering, lighting and shading

A 3D scene can be rendered in a number of straightforward ways:

- wire frame: display all edges
- hidden line removal - display only those edges and parts of edges which are not obscured by solid objects
- flat shading (or constant shading).

These have the attraction of being fast, but provide limited information about to the user about the scene. It is difficult to perceive the relative depths and gradients. We therefore need to think about lighting and shading. First a simple principle:

We see things, that are not themselves light sources, by the action of light being reflected off them of transmitted through them.

Therefore we need to understand surfaces (and interiors) and lights.

4.4.1 Conventional model for surfaces and incident light

There is no real agreement on the most suitable surface and lighting model to use. Often the determining factors are pragmatic ones. We outline here a simplified model which is in common use.

We first observe that light can be modelled by considering

- ambient illumination, in which there is no variation in intensity of direction
- and a set of specific light sources. When light from any of these sources hits a surface, the direction of incident light is defined.

We next model the reflection from a surface by splitting it into these components. Firstly the response to ambient light depends on:

- a factor for ambient light reflection for this surface \(k_a\)
- a factor for diffuse reflection for each colour \(w\) for this surface \(O_{dw}\)
- the intensity of ambient light \(I_{aw}\)

The response to ambient light is: \(k_a O_{dw} I_{aw}\)
Secondly, the diffuse (rough) response to the specific light sources depends on:

- a factor for diffuse light reflection for this surface for each colour \((k_d)\)
- \(O_{dw}\) as for ambient light
- the intensity of incident light from each specific light source \((I_{sw})\)
- the angle between the direction of incident light and the normal vector to the surface \((\theta)\)

The diffuse response is: \(k_d O_{dw} I_{sw} \cos \theta\) summed over all light sources

Thirdly the specular (shiny) response to the specific light sources depends on:

- a factor for specular light reflection for this surface \((k_s)\)
- a factor for specular reflection for each colour \(w\) for this surface \((O_{sw})\)
- the intensity of incident light from each light source (the same as for the diffuse response \(I_{sw}\))
- for each light source, the angle between the viewing direction and the direction of the maximum reflection \((\alpha)\)
- a specular exponent for the surface, expressing the concentration of the reflected light about the direction of maximum reflection \((c)\)

The specular response is: \(k_s O_{sw} I_{sw} \cos^c \alpha\)

This specular model was one of Phong’s contributions (except for the \(O_{sw}\) term) - the Phong specular reflection model.

### 4.4.2 Shading interpolation

Often the application approximates a curved surface by a large collection of polygons. To calculate the reflected light, we need surface colour information (called the intrinsic colour in PHIGS PLUS). Also, when shading, we may want to simulate a curved appearance, so in that case we need surface normals everywhere.

Unfortunately if the curve has been approximated by polygons, the surface normal is either available only at the vertices or worse still has been discarded. In this situation, it is necessary to interpolate the normals across each polygon and, at each point where needed, use that to calculate the reflected light everywhere. This is normal vector interpolation or Phong shading.

Since the lighting calculation is expensive, an alternative is to calculate the colour at the vertices and interpolate colours everywhere across the polygon. This is usually much faster and is the principle behind Gouraud shading which is widely implemented in hardware. It is quite sufficient for many purposes, especially if there are no highlights. There can be some visual interference if the intensity changes across the object’s surface rapidly.
4.4.3 Ray tracing and radiosity

Algorithms described so far have gained wide acceptance in graphics hardware. However the solutions are far from perfect. One problem is that only single reflections are taken into account. The appearance of most real-life scenes is influenced by complex reflections and deeper algorithms are needed.

In outline, ray tracing algorithms are quite straightforward. A light ray emanates from a light source, it may hit one or more surfaces and may pass to the lens of the camera. However the number of light rays that never appear in the picture is enormous so this would be extremely wasteful.

Instead a ray is traced backwards. Given a pixel, we have to find which light rays contribute to it and what colour do they contribute? The fundamental operation is what object did this ray come from last, at what point on its surface was it incident and what is its angle of incidence.

Calculating the intersection of each object with each ray is time-consuming as there are many rays and (usually) many objects. Since the first ray-tracing algorithms, many improvements have been made, taking advantage of regularities in the scene or using parallel processing.

A full description of ray-tracing can be found in [7].

Since it is difficult to trace a ray backwards beyond a diffuse reflection, the method is suited to specular reflection, which is why these algorithms usually are usually seen producing highly polished surfaces. Unfortunately most real life scenes have a predominance of surfaces with a high diffuse component. Radiosity algorithms attempt to solve this.

A radiosity algorithm models all the light energy passing between the surfaces in a scene. It allows for light being reflected in many directions off a surface.

5. Summary

In this paper, an attempt has been made to introduce in straightforward terms some of the concepts of 2D and 3D computer graphics.

6. References


Virtual Prototyping at CERN

Silvano de Gennaro
CERN, Geneva, Switzerland

Abstract

This paper describes the activities of the VENUS project at CERN, whose task is to produce Virtual Prototypes of the Large Hadron Collider, and its experimental pits and detectors. These prototypes are made available to the CERN engineers on site, and to the whole HEP community through a 3D graphic WWW browser currently under development.

The VENUS Project

The VENUS [1] (Virtual Environment Navigation in the Underground Sites) project is probably the world's most extensive applications of Virtual Prototypes to Engineering design. Just over one year old, VENUS proposes the latest and most advanced graphic solutions to a number of problems connected with the design of LHC. In order to respond more promptly to the needs of LHC designers and engineers, VENUS features four subprojects:

- Virtual Prototyping
- Territory Impact Study
- Networked Design Integration
- Assembly Planning and Control

We will cover the methods used and results obtained in each of these fields.

Virtual Prototyping

Virtual Prototyping starts where CAD ends. The CAD objects designed by the CERN engineers are imported into a Virtual Environment, then optimised and treated for their surface properties (colour, material, texture, transparency etc.), and finally organised in a virtual world. The viewer is immediately able to fly through this world and explore it from the inside.

Virtual Prototypes are an ideal replacement for the wooden models traditionally built for the past CERN machines, as they are generated directly from the EUCLID CAD files, therefore they preserve the original accuracy, they can be updated in a matter of minutes, and they allow immersive visualisation, in any preferred scale, through any preferred navigation metaphor. Because the flythrough is performed using one of the numerous off-the shelf packages available on the Virtual Reality market, we are free to add any of the navigation peripherals supported by that package at no extra effort. At present the VENUS lab allows three navigation metaphors:

- Spaceball and computer screen (mono or stereo)
- joystick and video projector (mono or stereo)
- 3-D joystick and VR helmet (mono or stereo, immersive)

The Spaceball is a 6 degrees of freedom, coordinates and velocity input device, allowing easy displacement in any direction in a 3-D environment. This solution is ideal for developing models, as it preserves access to the keyboard, therefore to the package's X-windows interface. On-screen rendering offers higher image quality and can also be selected in stereo mode, using a pair of shutter glasses.
The second metaphor (joystick and video projector) is used for group visualisation. One person controls the flight through a joystick while a number of viewers can follow on a large stereoscopic screen, wearing polarised glasses. The stereo projector is installed in a conference room adjacent to the VENUS lab, where design work teams meet and discuss the development of the models. The large screen (160x160 cm) allows more detailed viewing and the stereo vision allows a better understanding of depth and volumes, particularly with semi-transparent or wireframed objects.

The third metaphor is the classic Virtual Reality approach to immersive navigation. A VR helmet places you inside the model, giving you a realistic perception of proportions. By moving your head around you can explore the virtual worlds as if you were really walking through it. Translations of your body in the virtual environment are controlled through forward and backward movement buttons on the 3-D joystick.

The LHC Virtual Prototypes are designed in EUCLID, then exported in Wavefront format to VENUS' SGI Onyx RealityEngine2, where they are visualised using Medialab's CLOVIS Virtual Reality package.

**Territory Impact Study**

The primary commitment of VENUS at present is to deliver accurate photorealistic 3-D reproductions of the landscape around the access pits that need new surface buildings, such as Point 5 and Point 1, which will host respectively CMS and ATLAS. The purpose of these models is to serve as a basis for a study aimed to minimise the visual impact of the future LHC surface sites on the territory.

The territory elevation curves, as well as the future surface buildings, are modelled in Euclid, then exported the same way as the Virtual Prototypes, to the Onyx. Here the land model is “dressed” with houses, fields, trees, road signs and other details, by pasting textures obtained by scanning photos and post-processing them on a Mac using Adobe Photoshop.

The result is a Virtual World that you can fly through and observe from any possible angle, using the software and metaphors described above. While moving you can also interact with the environment by moving the future buildings horizontally or vertically in real time, and observing the effect on the landscape, until you find the best solution. You can also add at a keyclick hills and forests, or watch the trees grow, in order to establish a planned landscape architecture over time.

**Networked Design Integration**

The LHC Virtual Prototypes are physically constrained to the VENUS lab, as they need exclusive hardware and software, available today only in a few other VR laboratories around the world. Therefore they are usable within the scope of the infrastructure design of the pits and the machine, which takes place nearby the VENUS lab, but unfortunately they cannot be made easily available to all the institutes who will design the detectors components.

In order to allow interactive world-wide access to the experiments' 3-D data, VENUS has committed to develop a specialised package to provide web-based, on-screen navigation. The first version of this tool is called i3D [2]. It has been written at CRS4 [3], in Italy, and is available free of charge on the public domain. VENUS will pursue the enhancement of this software in order to use it as a graphic integration tool for LHC.

I3d stands half way between a VR navigation tool and a web browser. As a VR tool it performs on line rendering and navigation, featuring also some of the VR peripherals, such as
Spaceball and stereo glasses. As a web browser it supports hyperlinks, allowing to download all kinds of objects supported by the web browsers. i3d worlds are loaded via http, in i3D's own format "3d" or in VRML format "wrl", "vrml". VRML [4] stands for "Virtual Reality Markup Language" and is a new standard for VR worlds description. When clicking on a "i3d", "wrl" or "vrml" within a web browser such as Mosaic or Netscape, one triggers the spawn of i3D in a separate window. i3D will then load its world and let you navigate with the look and feel of a VR product. Some of the graphic objects you encounter in the "flight" are bordered in blue. These are hyperlinks to other worlds, or to other web objects. For instance by clicking on a muon detector you may cause Netscape to load the home page of the institute where it is being designed, by clicking on a magnet you can get its technical drawing in a GIF file, and a click on an argon tank can load the whole cryogenics room in your world.

i3D is available at present only on Silicon Graphics stations. Future development of i3D will be carried out at CERN as from September. It includes porting to other UNIX™ stations and support for engineering features, such as an information window to obtain technical and metric data about the objects.

Assembly Planning and Control

Although VENUS can be defined mainly as a visualisation lab, its scope goes much beyond the mere visual world. One of the major needs of the engineers in charge of LHC is to plan the operations for the installation of the detectors infrastructures and equipment into the pits.

The ATLAS toroid will be preassembled in items weighing 200 tons each, CMS in three pieces of 1000 tons each. Most of these items must be moved at a maximum acceleration of 0.1g and must be aligned with a precision of 6 microns. The operation of lowering these parts into the pits cannot be improvised, but needs careful study and simulation. VENUS is currently evaluating a few software packages that could allow not only to simulate the operations, but also to perform them automatically. After converting our Virtual Prototypes to the standard accepted by these packages, we will be able to let the computer calculate for each item a collision-free path, within the degrees of freedom of our bridge cranes. This path could then be injected into a robot controller that will drive the cranes during the build phases. This should allow us to assemble our future detectors almost without human intervention, therefore minimising the probability of erroneous manoeuvres.

Acknowledgements

VENUS is a pilot project sponsored by the CN (Computer and Networks) and AC (Accelerators Direction) divisions, staffed by myself, plus three part-time members: Nils Høymir, Lars-Johan Rehn and Michele Müller. Many thanks to Damien Riunaud for spending hours scanning mountains, to GianVittorio Frigo for valuable consultancy on Photoshop, and to Chris Jones and David Williams for the enthusiastic support to this project.

References

Event Display: Can We See What We Want to See?¹

H.Drevermann  
*CERN, Geneva*

D.Kuhn  
*Institut für Experimentalphysik der Universität Innsbruck*²

B.S.Nilsson  
*CERN, Geneva, and Niels Bohr Institute, Copenhagen*

Abstract

Due to the complexity of HEP detectors and their data the graphical representation of events is necessary, but also very demanding. The paper covers physiological aspects, e.g. choice of colours, human pattern recognition and 3D vision. For the unbiased understanding of the data realistic 2D and 3D projections, schematic projections and abstract 2D and 3D projections are discussed.

1 Introduction

High energy physics experiments investigate reactions between colliding elementary particles. To this purpose data on the particles leaving the collision point are recorded in large detectors and stored in digital form. The set of data recorded per collision is called an event. Practically all subdetectors are sampling devices, which for each event record the tracks of charged particles as a sequence of points, called hits, or record the showers of particles as a set of cells, for which the energy deposited is recorded as well.

The events are the basic units for further investigations, which are done by powerful pattern recognition and analysis programs. For checking of these methods and for presentation, the display of single events is an efficient tool, as visual representation is the most efficient way to transfer data from a computer to the human brain.

However, complexity of both events and detectors has increased substantially and will increase further. Higher event multiplicities and higher momenta of outgoing particles can be matched by more sophisticated detectors, i.e. detectors with a growing number of subunits of increasing granularity, resolution and precision. As a consequence pictures of detectors and events are getting more and more complicated and, in the extreme, may even get incomprehensible. The enormous improvements of detectors, of computers and of visual devices seems not to be matched by the “perception techniques” of the human eye and the human brain, as these have been developed long time ago for other objects than pictures of events. This leads to the question: Is a fast, efficient and unambiguous transfer of data to the human brain via visual techniques still possible for complicated events or will it be more convenient to read and interpret numbers?

Conventional graphics tries to represent events in the most realistic way³. However, it will turn out for a variety of applications, that these conventional representations result in pictures, which are not sufficiently clear. Therefore new visual representations are proposed here, which can better be tuned to the capabilities of human perception.

¹ This in an extended version of "Is there a Future for Event Display?" [1]
² Supported by grant of Fonds zur Förderung der Wissenschaftlichen Forschung, Austria
³ Complicated events might be cleaned by selective presentation of parts of the data and the detector
For the use of graphical representations in talks and papers it is necessary to find pictures, which can be understood intuitively without omitting relevant information. For this purpose in this article we will discuss:

- the selection of clear views,
- methods to present histograms,
- coloring schemes.

For the checking of detector performance and of programming tools one needs independent methods. One of the best methods to fulfil this task is visual analysis, which normally can be applied only to a small subsample of the large amount of events recorded in a typical experiment. Visual analysis may even go beyond the capabilities of other methods in recognizing specific event features. It will be shown here, that there are ways of visual data presentation beyond the conventional 2D and 3D techniques, which facilitate these tasks considerably. We will discuss in detail:

- the selection of special representations,
- methods of picture transformation in two and three dimensions,
- the association of tracking information to data of scalar fields (e.g. Lego plot),
- representation of scalar fields in two and three dimensions.

Most of the techniques discussed here were developed and applied for the ALEPH experiment at LEP/CERN [2]. Their direct application to other experimental setups is restricted to cylindrical detectors with a homogeneous solenoidal field. However, it seems possible to modify these techniques for application to different setups. In some cases this will be done here through a generalization of the methods and subsequent application to other experimental devices, namely to tracking detectors outside a magnetic field or experiments without magnetic field.

All techniques mentioned here are incorporated in the graphics program DALI which runs on simple workstations.

2 Pictures for Talks and Papers

Pictures of events are often used in talks and papers to underline and clarify what is said, i.e. to show, what would be lengthy and difficult to explain in words. Such pictures must be easily and intuitively understood, requiring normally only short explanations. The listener or reader should be allowed to assume that his intuitive understanding of the picture is right. It is the speaker’s or writer’s responsibility to guarantee, that the impression one gets from a picture is the right one.

2.1 Front and Side View

If no event is shown, a detector is best displayed in a technique called cut away perspective view. This resembles a photo of the real detector, where parts of it are cut away to show its interior. For the simultaneous representation of the detector and an event, however, this technique is normally not applicable. In this case the detector is drawn usually in what is called the wire frame technique. Figures 1a and d show perspective projections in this technique.

These pictures, however, are too crowded with lines and need to be simplified. This is done in the $Y$ versus $X$ projection ($Y/X$: $Y$=ordinate, $X$=abscissa) in figure 1b and in $Y/Z$ in 1e. The $Z$-axis is equal to the cylinder axis. Compared to the perspective projections, the number of visible lines in $Y/X$ and $Y/Z$ is typically reduced by a factor three.
Figure 1: Drawing of the barrel and endcaps of the HCAL(H), ECAL(E) and TPC(T) in the projections:

a) perspective  
b) Y/X  
c) Y/X without endcaps, with TPC sectors  
d) perspective without endcaps  
e) Y/Z  
f) ρ′/Z with inner detectors

These projections, however, suffer still from the fact, that different subdetectors are superimposed onto each other. This problem can be solved, if the projections Y/X and Y/Z are replaced by pictures of cross-sections through the detector center perpendicular (figure 1c) or parallel (figure 1f) to the Z-axis.

\footnote{In Y/X the ECAL and HCAL endcaps and the TPC overlap, in Y/Z the ECAL, HCAL barrel and the TPC.}
However, in such cross-sections events cannot be displayed in a useful way, since a cross-section through a line yields normally only one point. Therefore, one needs projections, which on one side preserve the line character of tracks and on the other side result in the same detector images as the ones obtained from cross-sections.

In the case of the ALEPH detector and of similar ones with cylindrical structure, the $Y/X$ projection with endcaps omitted is identical to the cross-section perpendicular to the cylinder axis (front view). This type of picture will be called $Y/X$ from now on.

The cross-section parallel to the cylinder axis (schematic side view) is identical to a $\rho'/Z$ projection, where $\rho'$ is defined as $\rho' = \pm \rho = \pm \sqrt{X^2 + Y^2}$ with the sign depending on the azimuthal angle $\left( \varphi = \text{atan} \frac{Y}{X} \right)$ of the object to be drawn; $\rho' = +\rho$, if $\varphi < \varphi - \varphi_1 + 180^\circ$ and $\rho' = -\rho$, otherwise, where $\varphi_1$ is interactively defined. In $\rho'/Z$ the event is cut into two unconnected halves (see figure 2c) and even single tracks may be cut into two pieces.

In the case of $Y/X$ as defined above, hits in the omitted endcaps are not drawn. For both projections, $Y/X$ and $\rho'/Z$, the following rules hold for the observer:

- hits lie inside the subdetector, from which they originate, so that their source is obvious. That means also, that
- hits or tracks are visible only if the corresponding subdetector is drawn.

These features, which facilitate interpretation considerably, are lacking in the other projections of figure 1.

The detector elements show up more clearly, if the background around and between them is shaded or colored (compare figures 2a,b to the figures 1c,f and see color plate 1). The presentation of subdetector areas in different colors is improved considerably by overlaying the wireframe, which facilitates the understanding of the structure of the subdetectors. The use of colors or gray shades gives a clear improvement as compared to the mere wireframe picture, especially if a detector section is shown (see color plate 1).

### 2.2 "Fish Eye Transformations"

In the case of radial symmetric pictures, as the $Y/X$ projection of a cylindrical or quasi cylindrical detector such as ALEPH, the scale may be decreased with increasing radius, so that the outer detectors appear shrunk. For a constant total picture size the inner subdetectors are hence enlarged (compare figure 3a and c). This emphasizes the commonly used construction principle of detectors, namely that precision and sampling distance decrease, when stepping from the inner to the outer detectors (see color plate 3).

Such a transformation, which we call "circular fish eye transformation", is calculated in the following way: From the cartesian coordinates $X$ and $Y$ the spherical coordinates $\rho$ and $\varphi$ are derived. These are transformed to $\rho_F$ and $\varphi_F$ by: $\rho_F = \frac{\rho}{1 + a\rho}$ and $\varphi_F = \varphi$. From $\rho_F$ and $\varphi_F$ the cartesian coordinates $X_F$ and $Y_F$ are recalculated and drawn with a suitable linear scale to conserve the total picture size. The factor $a$ is chosen interactively [3].

---

5 For the display of tracks going into an endcap, $\rho'/Z$ is the only good, intuitively understandable projection.
The non linear fish eye transformation gets linear for small $\rho$ ($a\rho \ll 1 \Rightarrow \rho \approx r_p$) avoiding a singularity at the origin.

![Figure 3: Comparison of linear projections to fish eye projections](image)

**Figure 3:** Comparison of linear projections to fish eye projections

- a) linear $Y/X$
- b) linear $\rho'/Z$
- c) circular fish eye transformation of $Y/Z$
- d) rectangular fish eye transformation of $\rho'/Z$

It transforms circles around the center into circles of another radius. Straight radial lines remain straight with the same angle (compare figure 4 a and b). If this transformation is applied to a picture of rectangular structure, as the $\rho'/Z$ projection in figure 3 b, one gets a picture, which is difficult to interpret, as horizontal and vertical lines get curved (compare figure 4d and 4e).
Figure 4: Fish eye transformations of circles and radial lines (a, b, c) and of vertical and horizontal lines (d, e, f)

a) linear  b) circular fish eye  c) rectangular fish eye

d) linear  e) circular fish eye  f) rectangular fish eye

The rectangular fish eye transformation of the $\rho'/Z$ projection:

$$\rho'_F = \rho' \cdot \frac{1}{1 + a\rho}, \quad Z_F = Z \cdot \frac{1}{1 + aZ}$$

(1)

does not transform horizontal lines into horizontal ones and vertical lines into vertical ones (compare d and f of figure 4). Circles get deformed towards rectangles and radial lines get curved (compare a and c of figure 4). This transformation yields the picture in figure 3d where the tracking detectors are better recognized.

2.3 Histograms in a Picture

Particles showering in calorimeters or just traversing them deposit energy. In order to represent the position of the cells and the deposited energy, the active cells of size $\Delta_1$, $\Delta_2$, $\Delta_3$ and their energy deposit $E$ are commonly displayed by representing them as boxes of size $\Delta_1$, $\Delta_2$, $kE$, i.e. the length of one side is replaced by the properly scaled energy $E$.

If a projection is chosen, in which the cells line up behind each other, one gets a picture as seen in color plate 2a which resembles a “wire frame skyline”. This representation differs from a histogram, where the energies of cells lining up behind each other are added. Different modes of presenting such histograms are shown in color plate 2b-2f as wire frame (color

---

6 For other detector geometries, analogue transformations can be found
plate 2b), structured wire frame (color plate 2c), unframed area (color plate 2d), framed area (color plate 2e) and structured area (color plate 2f). Experience shows, that histograms drawn as structured areas (color plate 2f) are preferred by the users. If drawn as wireframe only, the structuring yields a more complicated picture (color plate 2c) compared to the unstructured wireframe (color plate 2b).

If histograms are displayed in a picture of radial structure they are best drawn as radial histograms (see Color plates 2 and 3). Even so the detector image is rectangular in $\rho' / Z$, radial histograms underline better the radial event structure (see color plates 4 and 5).

It may occur, that histograms from different detectors overlay each other. In order not to lose relevant information, four methods may be applied:

• Scaling down of the histograms, in order to avoid overlapping.
• Drawing both histograms as wire frames (see color plate 4a).
• Drawing the first histogram as structured area and the second one on top but as wire frame only.
• Drawing both histograms as structured areas, but in the following sequence: histogram 1, then histogram 2, then the wire frame of both histograms (see color plate 4b, histogram 1 = white, 2 = yellow).

Experience shows, that the last method produces the clearest pictures, but necessitates two passes to draw the same data.

2.4 Application of Colors

The choice of colors depends primarily on the size of the objects to be drawn and on the background, which they are drawn onto. The width of hits and tracks, i.e. points and lines, should be kept sufficiently small, in order to resolve them properly. However, in the case of small objects the human eye distinguishes only very few colors. In other words, it is the number of different colors requested for a display, which defines the object size. A good compromise is the use of the following colors: white or black, green, yellow, orange, red, magenta, blue, cyan and grey (see color plate 6a).

This reduced set of colors is in most cases not sufficient to convey quantitative information (e.g. the depth of an object), i.e. it cannot be used as representation of a third dimension. Colors are however very useful to identify objects on different pictures side by side (see in color plates 5 and compare to color plate 3). This method can even be extended to lists, thus combining pictorial and numerical information. Track separation and association in different views via color is improved, if similar (e.g. close) objects have different colors (see color plates 3 and 5).

2.5 Definition of colors

Objects must be displayed in a way, that they are easily detected and their color is easily identified. For this purpose the display colors of objects and background must be carefully selected.

Any color on a display is composed out of three colors, namely green, red and blue. The contribution of each of these three colors is defined by three independent fractions $F_g$, $F_r$, $F_b$, which represent the intensity of green, red and blue relative to their maximum value. Display screens are normally adjusted such that one gets black, grey or white for $F_r = F_g = F_b$. 
The basic display colors are defined in the table below:

<table>
<thead>
<tr>
<th>color</th>
<th>black</th>
<th>blue</th>
<th>red</th>
<th>magenta</th>
<th>green</th>
<th>cyan</th>
<th>yellow</th>
<th>white</th>
</tr>
</thead>
<tbody>
<tr>
<td>abbrev.</td>
<td>K</td>
<td>B</td>
<td>R</td>
<td>M</td>
<td>G</td>
<td>C</td>
<td>Y</td>
<td>W</td>
</tr>
<tr>
<td>Fg</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Fr</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Fb</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

| Color diff. | B | R-B | B | G-R-B | B | R-B | B |

**Table I**: Basic display colors

If tiny objects of color $Fg_o$, $Fr_o$ and $Fb_o$ are displayed on a background of color $Fg_b$, $Fr_b$ and $Fb_b$, and if the three fractions of one of the sets are increased or decreased continuously, one gets to a point where the objects are hardly visible on the background, because both colors are seen with equal brightness. To avoid this, sufficiently high brightness contrast is necessary for easy object identification [7].

If we compare green and magenta as described in the paragraph above, we find that the brightness of green is substantially higher than the brightness of magenta = red + blue. In the same way we find that red is brighter than blue and yellow brighter than cyan. As colors get brighter if one of its fractions, $Fg$, $Fr$ or $Fb$, is increased, the above observations allow to order the basic colors by their brightness. In the table above the brightness of the basic display colors increases from left to right. Ordering the display colors through the well known color circle does not allow to judge the brightness of colors.

The above table and the experience that display colors differing by blue only are poorly distinguishable [8] leads to the following table which summarizes how well objects are separated from the background:

<table>
<thead>
<tr>
<th>white</th>
<th>yellow</th>
<th>cyan</th>
<th>green</th>
<th>magenta</th>
<th>red</th>
<th>blue</th>
<th>black</th>
</tr>
</thead>
<tbody>
<tr>
<td>white</td>
<td>-</td>
<td>bad</td>
<td>fair</td>
<td>fair</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
</tr>
<tr>
<td>yellow</td>
<td>bad</td>
<td>-</td>
<td>fair</td>
<td>fair</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
</tr>
<tr>
<td>cyan</td>
<td>fair</td>
<td>fair</td>
<td>-</td>
<td>bad</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
</tr>
<tr>
<td>green</td>
<td>fair</td>
<td>fair</td>
<td>bad</td>
<td>-</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
</tr>
<tr>
<td>magenta</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>-</td>
<td>bad</td>
<td>good</td>
</tr>
<tr>
<td>red</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>bad</td>
<td>-</td>
<td>good</td>
</tr>
<tr>
<td>blue</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>good</td>
<td>good</td>
<td>-</td>
</tr>
<tr>
<td>black</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>optimal</td>
<td>good</td>
<td>good</td>
<td>bad</td>
</tr>
</tbody>
</table>

**Table II**: Feasibility of separation and identification of basic display colors relative to each other

---

7 The same ordering is used in the test pictures of public TV channels
A second problem arises when one wants to distinguish two tiny objects with color $C_1$ and $C_2$. The table above can be applied here as well as a general guide line.

Furthermore, the separation of two objects by their color may depend on the background color. An example is given in a simple test, where a straight line is drawn, with a first part using color $C_1$ a second part using $C_2$ and a third part with $C_1$ again. For comparison a second line with $C_2$ should be drawn not too close to the first line. If $C_1$ is set to white and $C_2$ to yellow, the yellow appears more white on a blue or magenta background and, therefore, the middle part of the first line in yellow is hardly to identify on such a background. The same is true when setting $C_1$ and $C_2$ to cyan and green. This effect exemplifies that choosing an optimal background is not trivial.

Special problems, when setting up colors, turn up for printers. Except for its basic colors, which are often identical to the basic display colors, all other colors are produced through dithering, i.e. one recognizes a texture on low end printers. If tiny objects are drawn, picture quality is degraded if the visible size of the texture of objects and of background is not substantially smaller than the size of the objects. Only those parts of the background, where no objects are expected, may be drawn using non basic colors.

### 2.6 Use of Frames

The color plates 6a,b,c show points of varying size on black, blue and white background. For the representation of small points, a rather dark background is preferable. However, a light background is often preferred for a variety of reasons. Experience shows, that visibility of points is enhanced substantially, if points are surrounded by a thin frame (see color plates 6 d,e,f). The color of small points surrounded by a white frame and drawn onto dark background is often difficult to recognize, as the points seem to be white. For large objects however, a white frame improves recognition considerably, e.g. for blue objects on black background.

Due to the frame, the effective size of points or lines increases, which leads to a loss of resolution. This is overcome by drawing first all frames and in a second pass all points and lines. This is demonstrated in color plate 7 with a blow-up of 4 tiny points drawn on black and white background without frame (a), drawn sequentially, i.e. frame, point, frame, point... (b), and drawn in two passes (c). Experience shows, that resolution is not decreased in this way, but framing deteriorates recognition in regions with a high density of data, e.g. the inner (black) detector in color plate 3.

The methods described until now lead to a picture as seen in color plates 3 and 5, which are fairly easy and fast to understand.

### 3 Visual Analysis

In most experiments a large number of events are stored. It is often necessary to examine visually a subset of these events for a variety of reasons, as e.g.:

- check of hardware and software (on- and off-line)
- check of selected events,
- intuitive understanding of events and of possible problems,
- search for suitable events for talks and papers.
The examination should be effective and unbiased. However, one cannot assume that the intuitive impression one gets from the picture is right, in contrast to the situation, where one is looking at a picture presented in a talk or paper. One may be mislead for several reasons:

- loss of information,
- false assumptions,
- suggestive results.

In short: what looks good, may be wrong.

In the following we will discuss a variety of different representations, which help to avoid misinterpretations. Starting with representations of two dimensional data we will concentrate then on how to represent three dimensional data.

### 3.1 Use of picture transformations

It is common use in physics, particularly when handling multi-dimensional data, to apply suitable non-cartesian projections in order to better visualize the data, e.g. transverse versus longitudinal momentum, $P_T/P_Z$, which corresponds to $\rho/Z$ in coordinate space. The choice of these projections depends strongly on the data to be displayed. A well known example is the application of logarithmic scales. It may be regarded as a non linear picture transformation and is particularly powerful for the examination of exponential curves, which are linearized (compare figures 5a and d) taking advantage of the fact that human perception can better judge and extrapolate straight lines than curved ones. In contrast to the original picture in figure 5a, the change of exponent is clearly visible in the logarithmic representation of figure 5d.

![Figure 5: Linearization of an exponential curve and of a segment of a circle](image)

a) exponential curve  
b) segment of a circle  
c) compressed $\varphi/\rho$  
d) same curve as (a) in log. scale  
e) same segment as (b) in $\varphi/\rho$  
f) linear transform. of $\varphi/\rho$
It will be shown below that segments of circles (figure 5b) can be linearized by a \( \varphi/\rho \) projection (figure 5e). Through subsequent linear transformations it is possible to enhance features, which are otherwise difficult to extract (see the kink in figure 5f and c). These and similar methods will be discussed in the following.

3.2 Helices in Cartesian and Angular Projections

Due to the radial event structure and the cylindrical detector structure it is of interest to investigate the use of angular projections, i.e. of projections based on cylindrical and spherical coordinates\(^8\). The \( \rho'/Z \) projection discussed above may also be regarded as an angular projection.

In many detectors, such as the LEP detectors, tracks of particles are recorded, which move in a homogeneous solenoidal magnetic field parallel to the Z-axis. These tracks are described by helices. In order to better understand the use of angular projections, the helix equations will be formulated in cylindrical and spherical coordinates.

Neglecting multiple scattering, charged particles of momentum \( \hat{P} = [P_x, P_y, P_z] \) passing through a solenoidal field move along helices. Assuming that they start from the collision point at the origin of the coordinate system, the helices may be described in cartesian and in spherical coordinates as function of \( \Delta \alpha = 2\Delta \varphi \) in the following way:

\[
\begin{align*}
X &= cP_T \left[ \cos (\alpha_0 + \Delta \alpha) - \cos \alpha_0 \right] \\
Y &= cP_T \left[ \sin (\alpha_0 + \Delta \alpha) - \sin \alpha_0 \right] \quad \text{with} \quad \alpha_0 = \varphi_0 + 90^\circ \\
Z &= cP_Z \Delta \alpha \\
\end{align*}
\]

\[
\begin{align*}
\tan \varphi &= \frac{Y}{X} \\
\rho &= \sqrt{X^2 + Y^2} \\
R &= \sqrt{X^2 + Y^2 + Z^2} \\
\tan \vartheta &= \frac{\rho}{Z} \\
\end{align*}
\]

\[
\begin{align*}
\tan \varphi_0 &= \frac{P_y}{P_x} \\
P_T &= \sqrt{P_x^2 + P_y^2} \\
P &= \sqrt{P_x^2 + P_y^2 + P_z^2} \\
\tan \vartheta_0 &= \frac{P_T}{P_Z} \\
\phi &= \varphi_0 + \Delta \varphi \\
\rho &= 2cP_T \Delta \varphi \\
R &= 2cP \Delta \varphi \\
\vartheta &= \vartheta_0 \\
\end{align*}
\]

the above approximations are valid for \( \sin \Delta \varphi \approx \Delta \varphi \). Most particles have a sufficiently large momentum to justify this approximation, i.e. their track radius is sufficiently large.

Figure 6a shows 6 tracks in \( Y/X \), where helices give circles, the radius of which is proportional to \( P_T \). The \( Y/Z \) projection of the same tracks leads to cycloids as seen in figure 6b.

It can be seen from the equations (3), that as long as \( \sin \Delta \varphi \approx \Delta \varphi \), helices are linear in the angular projections \( \varphi/Z \), \( \varphi/\rho \) and \( \varphi/R \) (figure 6d, e and f). Their inverse gradient is proportional to \( P_Z \), \( P_T \) and \( P \) respectively. In \( \rho'/Z \) (figure 6c) they are straight. In projections where any variable is drawn versus \( \vartheta \), e.g. \( \varphi/\vartheta \) (6g), one gets approximately straight, vertical lines. The approximation fails for helices which do not pass through the center and for helices with many turns. As long as a helix with many turns passes again through the Z-axis, it is

\(^8\) The most famous of such projections is the Mercator projection, which deals with the spherical structure of the earth.
described in $\phi/Z$ by a set of parallel straight lines (see track 2 in figure 6d). Particle momentum and charge cannot be estimated from the projections $Y/Z, \rho'/Z$ and $\phi/\theta$.

**Figure 6:** Helices in cartesian and angular projections

- a) $Y/X$
- b) $Y/Z$
- c) $\rho'/Z$
- d) $\phi/Z$
- e) $\phi/\rho$
- f) $\phi/R$
- g) $\phi/\theta$

The $\phi/\rho$ projection is particularly useful to extrapolate tracks into the barrel part of calorimeters, whereas $\phi/Z$ is the best projection for extrapolation into the endcap part. This is due to the fact that in projections of any variable versus $\rho$ the barrel parts of different detectors are separated on the picture, whereas in anything versus $Z$ the endcaps are separated, so that the rules of chapter 2.1 can be applied. $\phi/Z$ is preferable to $Y/Z$, as the longitudinal momentum $P_Z$ and the charge of the particles can be estimated.

### 3.3 Lines through Sequences of Points

In the pattern recognition programs the hits belonging to a track are searched for and a helix is fitted to them\(^9\). In color plate 8a three helices are drawn suggesting the existence of three independent tracks. The helices were derived from a fit to the hits seen in color plate 8b. However, the picture shows that two of the tracks belong together, as incoming and outcoming track from a decay of a charged particle, yielding a so called “kink”. Color plate 8c shows in another example a set of hits, which were joined by the pattern recognition program to two

\(^9\) In reality the fit may take into account multiple scattering leading to a curve which is only approximately a helix.
tracks, shown as lines. Although this assignment looks very convincing, it turns out to be less obvious, if only the hits are drawn (color plate 8d). A better way to show the hits together with their track assignment while avoiding the suggestive force of lines is to color the points according to their track assignment (color plate 8e).

The examples above demonstrate, that it is necessary to recognize tracks from their hits only. Therefore we need to understand how human perception connects a sequence of points to lines.

Figure 7: Track recognition by the human brain

To this purpose figure 7a displays 12 straight lines and in figure 7b the same lines formed of a sequence of points. In the blow-up of the crowded center region of figure 7a seven lines are easily distinguished. However, if the lines are drawn as a sequence of points and blown up (7b), the seven vertical lines are hardly identified. If, however, the picture is compressed in the direction of the lines (see bottom of figures 7b), the lines are easily identified, even if only drawn as points.

Figures 7c and d show the same 3 straight lines of rather different direction and length in a clean (7c) and a noisy (7d) environment, where in the latter case the two long lines with large spacing between neighbouring points are lost. Compressed pictures are shown at the bottom of these figures. In the noisy environment, the line in the direction of the picture compression is easily identified, contrary to the two other tracks, of which the long one was not compressed and the short one was “over compressed”.

One learns, that human perception identifies a line from a sequence of points by joining close points together and not by following - like a computer - a predefined mathematical function.

In the case of events of radial structure, there is no preferred direction of compression if one wants to visualize the total event (see figure 8a). However, there are methods of radial compression, e.g. the fish eye view (see figure 8b). The principle of such methods can be summarized as follows: the angle under which a point is seen from the center remains unchanged, but its distance from the center $\rho$ is changed via a suitably chosen function to $\rho_{NEW} = F(\rho)$, e.g. $\rho_{NEW} = a + b\rho$ with $a > 0$ and $0 < b < 1$.

![Figure 8: 2\pi compression of a total event in the ITC(I) and the TPC(T)](image)

A more powerful method consists in “unrolling” the picture to the $\varphi/\rho$ projection (see figure 8c) and compressing it (see figure 8d) in $\rho$ direction.

### 3.4 Blow-up of a sequence of points

In many detectors hits are recorded with very high precision. In the ALEPH TPC hits are recorded with a sampling distance of $d = 60 \text{ mm}$ and a precision of $\Delta = 180\mu$. In order to visualize errors of this size on a screen, one must blow up the interesting part of a picture such that 1 pixel corresponds to $180\mu$ at least. As a consequence a screen image with typically $1000 \times 1000$ pixels covers only a detector area of $180 \times 180 \text{mm}^2$ in the case of a symmetric magnification ($\text{aspect ratio} = 1$). Therefore one ends up with only very few points on the
screen, i.e. one looses the relevant context to all the other hits of the track. This gets even worse for detectors of higher precision.

If one is only interested in the errors perpendicular to the track direction, a small magnification in track direction and a high one perpendicular to it (aspect ratio > 1) yield a picture, on which many hits are visible as well as their deviation from a smooth track. Color plate 9a displays a section of Y/X with aspect ratio = 1. The rectangle shows a section, which is blown up to give the picture in color plate 9b with an aspect ratio defined by the sides of the rectangle.

However, if tracks are rather curved, they can only be contained in a correspondingly large rectangle, which means, that the magnification perpendicular to the track is limited. This can be overcome by first linearizing the track using $\phi/\rho$ (color plate 9d) followed by a sufficiently large linear transformation. As an example, the parallelogram\footnote{Using the rubber band technique, the parallelogram and the rectangle are defined interactively in any direction.} containing the tracks as seen in color plate 9d is transformed to the full size of the square picture (color plate 9e), where the scattering of the hits is now clearly visible.

A magnification can be reached (color plate 9e and f), which yields a picture similar to a residual plot (color plate 9c, upper part). The residual plot, however, has the disadvantage not to show hits of other tracks or isolated hits close by. Note the use of colors to associate the hits between color plates 9c and f.

In essence it turns out that through such methods the limits due to the resolution of the screen and of the human visual system can be overcome.

### 3.5 Visualisation of the Vertex Region and the Inner Detectors

It is sometimes required to blow up track images near to the vertex region, e.g. for the investigation of secondary vertices close to the vertex. This can be accomplished by a linear blow-up or by use of the fish eye transformation (see chapter 2.2), which allows to visualize the tracks further away from the centre as well. Close to the vertex, i.e. in the region of interest, it yields a picture similar to a linear blow-up, so that distances and correlations can be estimated correctly.

### 3.6 Imaging Events from Fixed Target Detectors

In many detectors, particularly in fixed target detectors, particles leave the interaction point in a preferred direction. Two questions, arising when visualizing such events, will be discussed here:

- How to estimate visually, by which amount straight tracks point beside the interaction point? This problem arises in experimental setups, where tracks are curved by a magnetic field but recorded in a detector further down outside the magnetic field.
- How to show simultaneously tracks in small subdetectors close to the center and large subdetectors further downstream?

The top of figure 9a shows a simulation of hits recorded outside a magnetic field. The amount by which the tracks point beside the interaction point is a function mainly of the particle momentum, which is of main interest. However, due to the large distance of the hits from the interaction point it is rather difficult to estimate this quantity from the picture without relying...
on a pattern recognition program providing straight lines for backward extrapolation as in the bottom of figure 9a.

The tracks can be described by \( Y = a + bX \), where \( X \) is the downstream axis. As mentioned above, the offset, \( a \), depends mainly on the particle momentum and only very slightly on the track direction, unlike the gradient, \( b \), which depends on both. The equation can be rewritten as \( Y' = \frac{Y}{X} = \frac{a}{X} + b = aX' + b \). One sees that the non linear transformation

\[
X' = \frac{1}{X}, \quad Y' = \frac{Y}{X}
\]

transforms straight lines into straight lines. They are shown in \( Y'/X' \) in figure 9b. From the gradient \( a \) of these lines the particle momenta can thus be locally estimated even in cases where the center point and the hits cannot be displayed simultaneously.

By further application of linear transformations one can derive a more general formulation \( X'' = \frac{X}{1 + cX}, \quad Y'' = \frac{Y}{1 + cX} \), which again leaves straight tracks straight [3]. This transformation corresponds to the picture formed in our eye, when looking with a grazing view onto a flat image. Figure 9c shows a picture - \( X/Y \) - from two subdetectors of very different size with 4 straight tracks. In \( X''/Y'' \) (9d) the tracks remain straight, but can be resolved in both subdetectors, and the track segments can be connected by a straight line. This transformation can also be applied to curved tracks. If only slightly curved, the track images are practically identical to those obtained from the fish eye transformation discussed before.

\[\text{Figure 9: Non linear transformations of straight track segments}\]

\[\begin{array}{cccc}
\text{a) } Y/X & \text{b) } Y'/X' & \text{c) } X/Y & \text{d) } X''/Y'' \\
\end{array}\]

---

11The \( Y'/X' \) projection was properly scaled.

12If \( Y \ll X \), \( X \) can be replaced by \( \rho \) leading to the fish eye transformation discussed before.
3.7 Island Representation of Calorimeters

Calorimeters are composed of cells, in which energy is deposited either by the traversing particles or by their showers. These cells may be grouped together in layers surrounding the inner part of the detector (see the three layers of points in the outer ring shown in figure 10a). The optimal projection for the representation of these layers depends on the geometrical structure of the calorimeter. The electromagnetic and hadronic calorimeters in ALEPH have a projective structure, i.e. neighbouring cells of different layers can be grouped into towers, the axis of which points to the center of the detector, as well for the barrel as for the endcap. This suggests the use of $\phi/\theta$ to display single layers, as:

- barrel and endcap can be shown on the same picture,
- pictures of different layers have the same geometrical structure.

In figure 10a hits lying in a given solid angle are shown. The same hits are shown in $\phi/\theta$ for layer 2 and 3 in figures 10b and c, respectively. The amount of deposited energy per cell is proportional to the size of the squares inside the cell. In order to analyze the shower development between these two layers, the active cells - cells with deposited energy - in layer 2 must be compared to the corresponding ones in layer 3. This is facilitated considerably, if only towers are drawn, which have active cells in at least one of the three layers (see figures 10d,e,f) [4]. This yields an irregular, island-like substructure identical in all three images, which helps to correlate intuitively the cells in the different layers to each other.

Figure 10: Display of tracks and calorimeter data in $Y/X$ (a) and $\phi/\theta$ (b-f) (The area of the solid squares is proportional to the deposited energy).
The next problem is how to associate tracking information to the calorimeter information, i.e. tracks to showers. In \( Y/X \) shown in figure 10a, the track to shower correlation is not obvious, due to the missing \( Z \)-information and to the overlap of showers lying behind each other. If \( \varphi/\theta \) is used instead, two methods may be applied:

- A track fit is made to the tracks, and the entry point into the first layer is displayed. In this case one relies on a good track recognition and fit. The direction under which particles enter the calorimeter is not visualized.
- The hits of the TPC are superimposed in a \( \varphi/\theta \) projection onto the first layer (see figure 10d). This method is rarely used, as further information is needed for an unambiguous analysis. In this representation there is no information, whether the azimuthal angle \( \varphi \) increases or decreases for the tracks, e.g. whether the right track seen in figure 10d is associated to the island above or below. For this and other reasons it is necessary to use additional projections and reliable methods of track correlation between the different projections.

Particle momentum and charge cannot be estimated from \( \varphi/\theta \). This problem and the difficulties when applying the second method, are caused by the fact that only two dimensional projections are applied. In the next chapters possible solutions to represent the full three dimensional information will be discussed.

4 Three Dimensional Representations for Visual Analysis

Many (sub-) detectors record the position of hits in three dimensions. Here we will assume, that the errors of all three measured coordinates are sufficiently small, so that patterns of tracks or showers can be meaningfully visualized in any projection.

For the representation of such data we will try to find single pictures or picture sets, which allow to extract all relevant information. Several methods are used to solve this problem:

- perspective projections, sometimes called 3D,
- volume rendering, shading etc.,
- smooth rotations on appropriate 3D-stations,
- stereo pictures on appropriate stereo stations,
- technical drawings showing front, side and top views,
- abstract methods applicable for special sets of data.

The application of these methods for visual event analysis will be discussed in the following.
4.1 Perspective Projections.

Figure 11a shows a two dimensional wire frame projection of two objects, which we interpret as being three dimensional cubes. In doing so, we apply more or less strictly a set of assumptions, namely

- Straight lines on the picture are straight in space.
- Each line represents one line only, i.e. it is not composed out of several lines, overlaid or just touching on the projection.
- Lines parallel on the picture are parallel in space.
- Lines touching each other at a corner, touch each other in space.
- The angles between lines at the corner are angles of 90°.
- Several discrete line segments lining up on the picture do so in space.

As a consequence we see two cubes in figure 11a. But there exists an infinite number of three dimensional objects, which yield the same two dimensional image. One such object, for which the above assumptions are not fulfilled, is seen in figure 11b, from which the “cube” in figure 11a is obtained by a rotation of 20° around a vertical axis in the picture plane. If we see a picture like the one in figure 11a, we prefer automatically the most familiar solution. However, we can only be sure that this interpretation is right, if we know beforehand, that the above assumptions are valid, i.e. that we know beforehand, that we see two cubes.

Figure 11: Perspective projections of
a) two “cubes”  b) two “cubes” rotated by 20°
\[\text{c) TPC + ECAL} \]
d) TPC + event (left eye)  e) TPC + event (right eye)
\[\text{f) TPC + ECAL blown up} \]
If we know, that what we see is a cylinder, we can reconstruct a three dimensional cylinder in our brain from a two dimensional projection of it. Therefore, a two dimensional projection of a detector, of which we know what it looks like, will form a three dimensional image in our brain (see figures 11c,d,e).

However, if we look at points or at lines of the picture of an event, no one of the above assumptions is valid a priori. For an event there is no way to estimate the depth of points and tracks (see figure 11d), i.e. the picture of the event remains two dimensional. Furthermore, one may be misled by basing a decision on the preference of the simplest solution, which is even reinforced, if a two dimensional projection of hits and tracks is combined with a three dimensional detector image. An exception are points (e.g. confirmed end points of tracks) from which we know, that they lie on a plane. If this plane is surrounding a volume, one gets two solutions, one for the front and one for the back plane.

The advantage of showing the detector lies in the fact, that the direction, from which we are looking, is visualized. Furthermore one transmits the information, that the data are three dimensional.

A problem arises, if a part of the picture is blown up in a way that the lines, which compose the cubes, the cylinders etc., are unconnected. In this case the assumptions above, even if valid, cannot be applied, so that a rather complicated picture results (see figure 11f).

In the case, that the event and not the detector is of main interest, we can conclude that, what is often called 3D, is in reality a 2D projection not able to convey all relevant information. There are classical methods to improve this situation, namely the combination of perspective projections with

- shading and volume rendering,
- smooth rotations,
- stereo pictures.

They will be discussed in the following.

### 4.2 Shading and Volume Rendering

The picture of an event consists of

- points and lines, representing hits and tracks, which must be kept thin, in order not to loose resolution for the display of many of them, and of
- rather small boxes representing calorimeter cells, where often cells lie behind each other.

In the first case volume rendering techniques cannot be applied as we cannot distinguish enough intensity grades or colors on small objects. In the second case the wire frame technique must be applied if cells should not be obscured. This excludes shading and volume rendering except that very few hits and tracks are drawn.

If shading and volume rendering are applied to several subdetectors surrounding each other, one is lead back to cut away perspective views, onto which a two dimensional projection of an event can be overlaid, but the hits or showers of the event do often not correspond to the subdetectors they are drawn onto. One may find simple events which allow this, but for visual analysis, which must cope with all kind of events, these methods seem not to be applicable.

### 4.3 Smooth rotations.

One of the best methods to get additional information about three dimensional objects from a two dimensional display is a smooth rotation around an axis the direction of which is
sufficiently different from the viewing direction. During rotation the three dimensions of the data are mapped onto three independent variables, i.e. their horizontal and vertical picture position and the speed of displacement.

When applying a small smooth rotation, it is possible to identify hits, tracks or track segments, which are close to each other, as they move with similar speed. In many cases this is insufficient, as the lateral displacement of the projection of a point is generally much smaller than its depth in space.

If one rotates by 90° one gets a smooth transition between two distinct views, which helps to associate the images of hits and tracks on the first view with their corresponding images on the second one. However, for high multiplicity events, this method of association becomes rather tedious. It will be shown in chapter 4.5, that three orthogonal views are needed in more complicated cases, i.e. rotations around different axes are required. Thus a thorough check of an event becomes time consuming and requires a fair amount of discipline from the operator. A special case is treated in chapter 5.

4.4 Stereo Images

If we one looks at the data presented in one way or the other in real 3D, e.g. on a stereo device, one understands better the detector picture and the event structure and one can confirm the assumptions of chapter 4.1. The display of the detector image helps considerably, as it gives the necessary information of the depth scale. As each of our eyes is a 'two dimensional image recorder', smooth rotations help us further to match the points and lines from the two images in our brain.

However, due to the relatively small eye distance as compared to the distance between our eyes and the objects, depth recognition, when looking at things, is considerably worse than lateral recognition. Therefore, we are unable to judge the curvature of a track, if it is curved in a plane through our eyes. These observations are easily confirmed with real objects.

From the figures 11d and e one gets a stereo image, if one succeeds to look at the left picture with the left eye and the at the right picture with the right eye. By comparing the two images one may estimate the tiny differences, which lead us to recognize a stereo image.

4.5 The Method of Technical Drawings

A commonly used method of representing three dimensional objects is found in technical drawings, where three orthogonal projections are used, normally the top, front and side view. In figures 12a, b, c five specially selected tracks are shown in the top (Y/Z), front (Y/X) and side view (X/Z). Apart from a small change in length, the group of three tracks (2,3,4) gives the same image in the top and front view, whereas they look very different in the side view. Contrary to them, the curvatures of tracks 1 and 5 are similar in the front and side view, but look different in the side view.

For a thorough check of a track - or a set of nearly parallel tracks - two projections are required, one where the track is seen curved and one where it looks practically straight. In the case of an event, where tracks go into many directions, at least three projections are required, as in the example above. This means that the event must be rotated around several axes.

However, the front view (Y/X) (see figure 12b) and the schematic side view, i.e. the non linear $\rho'/Z$ projection, (figure 12d) show all tracks from the side with the maximum and minimum apparent curvature, respectively. Therefore, the top, front and side view can be
replaced by the two "orthogonal projections" $Y/X$ and $\rho'/Z$ for tracks pointing roughly to the center.

Very close to the center, the radial event structure is normally broken due to measuring errors or possible secondary vertices, so that the interpretation of $\rho'/Z$ becomes difficult. It should be further kept in mind, that momentum conservation cannot be checked in this projection.

Even the use of two projections instead of three suffers from the fact, that one has to associate hits and tracks in one picture to their image in the other one. Therefore, it will be tried in chapter 6 to show ways of presenting three dimensional data in one picture only, using unconventional but less intuitive projections.

![Figure 12: Top, front and side view and $\rho'/Z$ of five selected tracks](image)

- a) top view ($Y/Z$)
- b) front view ($Y/X$)
- c) side view ($X/Z$)
- d) schematic side view $\rho'/Z$

For all methods of 3D data representations discussed so far, it is difficult to find solutions of simultaneous data compression of the total event, as discussed in chapter 3.3 for 2D representations.

### 5 Track Visualisation Close to the Primary Interaction Point

A special problem arises when examining secondary vertices near to the primary interaction point. Secondary interactions may manifest themselves by tracks crossing or nearly crossing at

\[ \sin \alpha = \frac{Y'}{\rho'} \approx \frac{Y'}{\rho} \]

This is due to the fact that a track looks straight in $Y'/Z$ with $Y' = Y \sin \alpha + X \cos \alpha$, if $\alpha$ is chosen such that the track lies approximately in the picture plane, which means $Y' \approx \rho'$. 
a point different from the primary interaction point. The distance between these two points is of main interest for physical analysis.

The spherical character of events vanishes close to the interaction point due to non perfect track extrapolations down to the center and due to possible secondary interactions. As a consequence, the angles under which the track elements are seen from a vertex may vary considerably so that angular projections - including $\rho'/Z$ - give pictures which are very difficult to interpret. A way out are conventional cartesian projections with varying view points.

On the other hand, analysis gets more simple, as sufficiently close to the center no hits are visible and the reconstructed tracks can be approximated by straight lines.

In the following some methods are described for close inspection of the vertex region.

5.1 Smooth Standard Rotations

In order to find a good viewpoint for inspection of the interaction region smooth rotations are commonly applied. A rotation is characterized by the rotation axis, position of the viewpoint and angular speed.

The choice of these quantities is facilitated by taking into account the structure of the interaction and the feature one is interested in. Good choices for a rotation axis relative to the interaction are position and direction

- of a selected single track,
- of a reconstructed track leading to a secondary vertex,
- of the "jet axis" through the interaction point, i.e. the direction of flight of a group of particles.

A good choice for the axis relative to the display system is to position it into the screen plane. With a fixed rotation axis which is not interactively changed, different values of magnification may be applied for the direction parallel to the axis and perpendicular to it, so that angles appear enhanced. In figure 13a,b the corresponding ratio was set to a value of 4.

![Figure 13: Crossing of tracks in space](image)

Figure 13: Crossing of tracks in space

a) tracks 1 and 2 parallel  
  b) rotation around track 1 by 90°  
  c) track pair distance

In order to determine the minimal distance between two tracks in space and to determine the points of nearest approach on the two tracks the following method can be applied:

1) select one of the two tracks as axis to lie in the screen plane (figure 13a track 1);
2) rotate around it until both tracks appear parallel in the projection (tracks 1 and 2 in figure 13a). Now the visible distance between the two tracks is equal to the distance of nearest approach of the two tracks in space.

3) rotate by 90 degrees. Now the points of nearest approach on the two tracks fall onto the visible crossing point of the tracks in the projection (figure 13b).

The method above is best understood by using a real mock-up of the problem. There seems to be no clear method for the corresponding problem, if more than two tracks are involved, as can again be verified using a mock-up. The only way out consists in checking separately all combinations of two tracks.

5.2 Artificial Display Cube

If one rotates lines, which run out of the picture on at least one side, we do not have the impression of a 3D rotation. The lines move up and down like wind screen wipers and one does not realize that the space position of the lines relative to each other is fixed.

This problem may be overcome by adding an "artificial display cube", ADC, to the picture as seen in figure 14a. The ADC is defined such that its edges are not parallel to the rotation axis. Its size is selected to be as big as possible without that the corners of the cube leave the window during rotation around the axis. In the “frozen” picture of figure 14a the ADC is larger. The ADC is not modified when zooming the tracks. The tracks are clipped at the faces of the cube (figure 14a). During rotation of the tracks together with the cube one gets a good impression of the 3D character of the tracks. This impression is mainly due to the clipping of the tracks at the faces of the cube.

Figure 14: Artificial display cube

   a) interaction near to the center       b) same interaction rotated by 90°

Without rotation the space position of an exit point is ambiguous for the observer, i.e. it may either lie on one of the three front planes or one of the three back planes. This ambiguity is resolved by drawing crosses at each exit point, with their arms parallel to the edges of the corresponding face and their size being smaller for larger depth of the exit point. During rotation
the flagging of the endpoints by the crosses further helps the visualisation of the three dimensional structure of the interaction.

The ADC can be larger, when the rotation axis is tilted relative to the plane of the screen, which can however not be done, if the exact position of the points of nearest approach of two tracks is visualised as described in the previous chapter.

In order to find an optimal frozen picture for presentation purposes, two different procedures are helpful:

• The interaction is rotated until an optimal presentation is reached. Then the ADC alone is rotated with continuous clipping of tracks and maximizing the cube size until the whole picture appears optimal.

• An optimal presentation of the ADC is chosen. Then, the tracks alone are rotated with continuous clipping of the tracks until the whole picture appears optimal.

Whereas 3D continuous rotations may be helpful for analysis, it is difficult to present such continuous rotations to a larger audience. The presentation of a frozen picture as in figure 14a may be misleading, as it suggests to the audience a certain interpretation of the picture, namely that tracks which touch on the picture do touch in space. This may not be true, as can be seen from figure 14b. Whereas the positions of the crosses are three dimensional, the depth of the inner track points cannot be estimated, i.e. their position remains two dimensional (see chapter 4.1).

The clipping of tracks at the sides of the ADC and the elimination of tracks outside of the ADC are not only helpful to improve the perception of a continuous rotation, but also improves stereo vision on an adequate display. The elimination of tracks outside of the ADC improves stereo perception.

5.3 Display of the Track to Track Distance

By use of continuous rotations or by fixed projections seen from well chosen viewpoints, it is simple to identify groups of tracks, which do NOT cross. But the contrary is not true, i.e. one cannot easily verify if several tracks come close to each other in a small volume.

A method to overcome the latter problem consists in displaying the space distance between pairs of tracks via the following procedure (example: see figure 13a,b):

1) choose axis relative to the interaction as described in chapter 5.1 (track 1)

2) combine pairs of forward tracks (1-2,1-3,1-4,2-3,2-4,3-4) and backward tracks separately (5-6)

3) calculate the distance between the track pairs in a plane perpendicular to the axis;

4) plot this distance versus the position of the plane on the axis, down to the interaction point (see figure 13c).

The tracks 1, 2 and 3 are candidates for tracks originating from a secondary vertex, since the corresponding track pair distances have a very low minimum at \( x = 0.85cm \) (figure 13c). This picture differs from all others in this paper as it combines tracks to pairs of tracks, whereas until now hits or tracks were drawn independently of the position of the other hits or tracks.

A vertical line was moved interactively to the crossing point of the projections of track 1 and 2 in figure 13b. The same line is then displayed in figure 13a and c to show the equivalence of the two methods applied.
6 The V-Plot, a Three Dimensional Representation for Visual Analysis of Tracks

It is the advantage of conventional projections, that they can be applied to a large variety of objects and experimental setups. However, this is also their biggest disadvantage, as it is difficult to optimize them for special problems. In the following we propose projections, which were specially developed for the ALEPH experiment. We will discuss a picture called V-Plot, which was developed for helices, i.e. tracks of particles moving in a homogeneous solenoidal field. We will then generalize the underlying principles and apply them to a different experimental configuration.

6.1 The Helix Representation via the V-Plot

It was shown in chapter 3.3 and figure 8d that tracks are better recognized in a compressed \( \varphi/\rho \) projection than in \( Y/X \). This is shown again in color plates 10a,b. The compression facilitates the identification of tracks, but not their separation, as the total picture space is reduced as well (color plate 10b). The best track separation is obtained via \( \varphi/\vartheta \) (color plate 10c). As discussed before, a representation in \( \varphi/\vartheta \) does not allow to estimate charge and momentum, in contrast to \( \varphi/\rho \), and it is not possible to verify, if tracks really enter and leave the chamber.

Therefore it is tempting to use a linear combination of the two projections, namely \( \varphi/ (\vartheta + k\rho) \). In color plate 10d, a slightly modified projection \( \varphi/ (\vartheta + kD) \) is shown with \( D = \rho_{\text{MAX}} - \rho \), where \( \rho_{\text{MAX}} \) is the outer radius of the tracking device, here the TPC. The value of \( k \) is interactively chosen and scales the gradient of the straight track images. This projection conserves most of the good features of both projections, namely approximately straight track images, the ease of momentum and charge estimation as in \( \varphi/\rho \) and the good track separation as in \( \varphi/\vartheta \).

\( \varphi/\rho \) and \( \varphi/\vartheta \) are two projections, which together represent the full 3D information of the data. They may be replaced by the two symmetric projections, \( \varphi/ (\vartheta + kD) \), introduced above, and \( \varphi/ (\vartheta - kD) \). If \( k > 0 \), they represent the full 3D information as well (see color plates 10e,f). As last step, the two projections are drawn on top of each other as seen in color plate 10g. The two superimposed track images of a single track form a \( V \) pattern, where the exit point, \( D = 0 \), lies at the tip of the \( V \).

A somewhat modified definition of the variable \( D \) is more useful: \( D = R_{\text{MAX}} (\vartheta) - R \), where \( R \) is the spherical radius defined above and \( R_{\text{MAX}} \) the distance of the outer surface of the tracking detector from the center in the direction of the hit, so \( D \) is the distance from the hit to the outer surface of the TPC in this direction. If the tracking detector is of cylindrical form, \( R_{\text{MAX}} \) depends only on \( \vartheta \), i.e. \( R_{\text{MAX}} = \min \left( \frac{\rho_{\text{MAX}}}{\sin \vartheta}, \frac{Z_{\text{MAX}}}{|\cos \vartheta|} \right) \), where the outer cylinder surface is given by \( \rho_{\text{MAX}} \) and \( \pm Z_{\text{MAX}} \).
The interpretation of this so called “V-Plot”, as can be derived from the equations (3) in chapter 3.2, is summarized in the following:

- **V-position**: \( \phi, \Theta = \text{spatial track direction} \).
- **V-direction**: \( \text{up or down} \Rightarrow \text{particle charge} \).
  
  (e.g. ALEPH: up \( \Rightarrow \) negative, down \( \Rightarrow \) positive charge)
- **V-angle**: \( \text{gradient} \approx \frac{1}{P} = \text{particle momentum}^{-1} \)
  
  wide V’s = high momentum,
  narrow V’s = low momentum.
- **V-width**: proportional to \( D \), i.e. to track distance from exit.
  
  The tip of the V denotes the track exit.
- **curved V-arms**: the track has either low momentum or its origin is outside the center.

This means, that one can retrieve the full 3D information from the V-Plot. This is due to the fact, that two projections are superimposed. As the ordinate \( \phi \) is the same for the two projections, a hit is represented by two points, which may be replaced by a horizontal line connecting them. Such lines have three degrees of freedom. Their center point gives \( \phi \) and \( \Theta \) and their length is a measure of the distance of the hit from the outer surface of the detector, so that one could in principle recalculate the three original hit coordinates. The V-Plot is therefore an abstract 3D representation of the TPC hits.

The V-Plot has a particularly high information content. However, one has to prove, that a human operator sitting in front of a terminal is able to work with these pictures, and especially, that the doubling of hits and of tracks does not give pictures, which are too complicated for visual analysis. Such an investigation cannot be done by theoretical arguments but by applying this technique to typical and to difficult events.

### 6.2 Application of the V-Plot, Example 1

To illustrate how to work with the V-Plot in practice, \( Y/X \), \( \phi/\rho \) and the V-Plot are compared in color plate 11 showing an event, which was cleaned by eliminating all noise hits, i.e. hits, which were not associated to tracks by the pattern recognition program. When stepping clockwise, i.e. with increasing \( \phi \), through the tracks, one can compare the track representations in \( Y/X \) (color plate 11a) with those in the V-Plot (color plate 11b). This comparison is simplified by passing through the compressed \( \phi/\rho \) projection in color plate 11c.

On color plate 11 several tracks are labelled by their measured momentum in [GeV/c] to demonstrate the relation between momentum and V-angle and to help the reader to associate the track images in the different windows.

Some special tracks are blown up in the inserts of color plate 11. The region around the track labelled “kink”, is blown up from the V-Plot (color plate 11h) and from \( Y/X \) (color plate 11e). The kink is very pronounced in the V-Plot when compared to \( Y/X \). As this kink is mainly due to a variation of \( \Theta \), it would be better visible in \( \rho'/Z \), which is not shown here.

The V-arms of the blue track in insert (color plate 11h) and the V-arms of the yellow track in color plate 11b are curved indicating that the tracks do not originate from the center.

The red V of the 1.5 GeV/c track is blown up in insert (g). The scattering of the points around a straight line is due to the limited precision of the measurements and multiple
scattering. This demonstrates that one reaches a magnification, where the detector precision gets visible, so that one is neither limited by screen resolution nor by the human visual system.

The region around the 3.7 and 8 GeV/c tracks (red and blue in color plate 11b) is blown up as \( Y/X \) (color plate 11d) and V-Plot (color plate 11f). The two tracks have one point in common, hence the two tracks cross in space, which cannot be unambiguously derived from \( Y/X \), as the depth of the hits is not represented. Many tracks crossing each other are found in \( Y/X \) (color plate 11a) but from the V-Plot (color plate 11b) one can derive, that no other tracks cross in space.

### 6.3 Application of the V-Plot, Example 2

A minor but rather helpful detail needs to be mentioned before discussing the next example. As \( \mathbf{\vartheta} = \arctan \frac{\rho}{Z} \) increases with decreasing \( Z \) for a fixed value of \( \rho \), left and right would be inverted when comparing \( \varphi/\vartheta \) with \( \rho'/Z \). To avoid this, the \( \vartheta \)-axis is defined as pointing to the left in the V-Plot and in all \( \varphi/\vartheta \) projections. This is indicated in figure 15a by the arrows.

![Figure 15: Use of the V-Plot](image)

Figure 15a shows the measured hits (noise not removed) for a typical event of the ALEPH TPC as V-Plot. The V-Plot provides a simple tool to select hits or tracks through a volume...
defined in position and size via $\phi$, $\theta$ and $\rho$. This volume can subsequently be visualized in other projections\textsuperscript{14} or again as V-Plot, as is shown in the following examples:

- The angular section defined by the large rectangle (15a) is shown as V-Plot in 15b, as $Y/X$ in 15c and as $\rho'/Z$ in 15d,
- the angular section defined by the small rectangle in 15a is shown blown up in 15f and as $Y/X$ in 15g,h,
- from the tracks seen in figure 15b, two are selected through the rectangle in this figure and shown as $\rho'/Z$ in 15e.

The kink in track 5, which is due to the decay of a charged particle, is enhanced through the vertical compression of the V-Plot (15b) as compared to the kink shown in $Y/X$ (15e).

The blow-up of the V-Plot (15b) shows a quadrilateral pattern typical for the decay of a neutral particle into a positive (2) and negative (1) particle. The corresponding tracks have a common origin in the TPC, i.e. away from the primary vertex. No other pattern of this form is found in 15a and 15b, i.e. no other decay of a neutral particle exists in the TPC. It is rather difficult to confirm that fact from $Y/X$ and $\rho'/Z$.

The assignment of hits to tracks by the pattern recognition program, as indicated through the lines in 15f and 15h, looks rather likely in the V-Plot, where it is checked in 3D, compared to $Y/X$, where the depth information is lost (see also color plate 8c,d,e). The probably false association of the hits to tracks by the program may be due to the fact that the two tracks cross in space, as can be seen from 15f.

6.4 Extrapolation of Tracks via the V-Plot

In chapter 3.7 the $\phi/\theta$ representation of calorimeter cells was discussed. However no satisfactory method was found to associate the tracking data from the TPC to the calorimeter data (see figure 10d). This problem is solved by use of the V-Plot.

Figures 16a and 16b show the front and side view of two tracks and those calorimeter hits, which lie in the same direction. (The same event is used as in figure 10). The $\phi/\theta$ projection of the three layers of active calorimeter cells is shown in figures 16d,e,f, where the shower development can be estimated easily, as discussed before. In figure 16c the V-Plot of the two tracks is superimposed to the calorimeter islands, so that the clusters created by the two charged tracks can be identified. If for one reason or the other the last hits of a track are missing (which is not the case here), the exit position of the track can easily be deduced, i.e. even in this case tracks and showers can be correlated.

As seen in this example, the V-Plot is that representation of tracking data which is complementary to the most used display of calorimeter data, namely the Lego Plot.

The V-Plot is also applicable to backward extrapolations of tracks into inner tracking subdetectors, e.g. into a vertex detector, as long as three dimensional data are recorded or in the case that the hits are confined to few layers, which are separately investigated. In the latter case, the variable D defined in chapter 6.1 should be redefined such that the tip of the V points to the selected layer. However, backward extrapolation further downwards via the V-Plot may lead to very complicated pictures, when getting too close to the vertex. This is the case if tracks do not point precisely to the vertex due to measuring errors, secondary decays etc., so that the variation of $\phi$ and $\theta$ gets very large.

\textsuperscript{14}The angles are interactively defined by use of a rubber band cursor, the values of $\rho$ is preset.
6.5 The V-Plot for Super High Multiplicity Events

Figure 17a shows the front view, $Y/X$, for a simulated “super event” with 221 tracks. The simulation was accomplished by superimposing many events from the ALEPH TPC. Only few tracks can be partly identified in the front view even when blown up (see figure 17b). The display of such “super events” in other conventional projections yields even more difficult pictures due to cycloidal track patterns. The simultaneous use of two conventional 2D projections, in order to examine the tracks from several sides, excludes itself, as it is practically impossible to correlate the tracks. These projections can only be used, if one succeeds to apply cuts so that sufficiently few tracks are displayed.

Track identification is possible via the V-Plot (figure 18a), as long as the tracks leave the chamber, i.e. do not spiral. By blowing up crowded regions (the framed region in 18a is blown up in 18b) practically all non spiralling tracks can be identified.
Figure 17: Super high multiplicity event with 210 tracks
   a) Y/X  
   b) blow-up of Y/X defined by the rectangle in figure a

Figure 18: Super high multiplicity event with 210 tracks
   a) V-Plot  
   b) blow-up of the V-Plot  
   c) Y/X of the hits selected through the rectangle of figure b.
By defining a volume through an angular section, as discussed in chapter 6.3, single tracks or groups of tracks can be selected and displayed in conventional projections. The hits contained in the rectangle of figure 18b are shown in Y/X in figure 18c, where tracks are easily recognized. This track selection acts like a depth cut (ϑ-cut, not Z-cut!) on Y/X. Despite the high multiplicity of such events it may be possible to generate conventional pictures suitable for talks and papers by applying selections of this type.

6.6 Generalisation of the V-Plot

The V-Plot technique is constrained to 3D data only, which might either consist of 3D hits or of 3D track segments. In the latter case the spatial position may be measured directly or may be determined through the averaging of sets of hits or may be obtained through other methods.

From the special form of the V-Plot, described above, we will try to deduce general rules to construct a V-Plot, namely:

• The V-Plot is a superposition of two symmetric projections. For each 3D hit two points are drawn. The position of the center between them and their distance kD are calculated from the coordinates of the hits. The angle ϑ under which the two points are drawn can be chosen in a convenient way. It is important that no other information than the 3D position of the hit is used. Especially, the association of a hit to a track is not taken into account.

• The projections must be chosen in a way to (approximately) linearize tracks and to compress all of them simultaneously. An optimal compression is achieved, if for k=0 a radial straight track is seen as a single point.

• The distance kD between the two point images is a measure for the distance R of the 3D hit from the center.

• H,V and D should be chosen in such a way that charge and momentum can be estimated. If a variable U (or a function of variables) is (approximately) constant for all points of a track, then H,V,D and ϑ may subsequently be modified through this variable in order to fulfil the above requirements and to yield a usable picture, e.g. H may be changed to H' = H ⋅ F(U). Under this transformation straight V-arms remain approximately straight.

Various realizations of V-Plots are published elsewhere [5]. In the next chapter one of them is presented.

6.7 The V-Plot for Straight Track Sections Outside a Magnetic Field.
(TPC Tracks of the NA35 Experiment)

In the fixed-target heavy-ion experiment - NA35 - at the SPS/CERN a large number of particles leaves the target in very forward direction, defined as X-direction [6]. A perpendicular homogeneous magnetic field (in Z-direction) bends the particle trajectories in the X-Y plane. About six meters downstream, outside the magnetic field, straight track segments are recorded by a TPC, delivering 3D track hits. Figure 19a shows the setup in Y/X, with the vertex at the very left and a rectangular block of the TPC hits at the right. A blow-up of the hits is shown in figure 19d and the other two projections Y/Z and Z/X in figures 19b,c respectively. The only projection, where tracks can be distinguished, is Y/Z. However, it is difficult, if not impossible, to estimate charge and momentum.
Figure 19: An event from the NA35 TPC
  a) setup of the experiment in Y/X
  b) Y/Z
  c) Z/X
  d) Y/X

Figure 20: Simulated hits of tracks with predefined momenta P (without measuring errors)
  a) $P=2$ GeV/c, Y/Z
  b) $P=6$ GeV/c, Y/Z
  c) $P=12$ GeV/c, Y/Z
  d) $P=2$ GeV/c, V-Plot
  e) $P=6$ GeV/c, V-Plot
  f) $P=12$ GeV/c, V-Plot
The straight tracks in the TPC can be described by \( Y = aX + b \) and \( Z = cX \), where the gradient \( a \) depends on the direction and the momentum of the track; the offset \( b \) is an approximate measure of the track momentum, hence is of higher interest.

Figure 21: V-Plot of the same NA35 event as in figure 19

The general V-Plot rules of the previous chapter are fulfilled by setting

\[
H = \frac{Z}{X}, \quad V = \frac{Y}{X}, \quad D = X_1 X_2 \left( \frac{1}{X_1} - \frac{1}{X_2} \right) = X - X_1, \quad \text{where } X_1 \text{ and } X_2 \text{ define the position of the entry and the exit plane of the TPC, respectively. Then one displays } V \text{ versus } H \pm kD.
\]

If \( k=0 \), this V-Plot is identical to \( \frac{Y}{X} \frac{Z}{X} \), which is rather similar to the best of the above projections, namely \( Y/Z \) (figure 19b); the interpretation of the V-position is therefore straightforward. In \( \frac{Y}{X} \frac{Z}{X} \) the image of a radial straight tracks \( (b = 0) \) is reduced to one point. For \( k \neq 0 \) the V-arms are drawn in the symmetric projections \( V/(H+kd/2) \) and \( V/(H-kd/2) \). For the points of a single track \( (Z/X = c) \) these projections are identical to \( \frac{Y}{X} \frac{1}{X} \) apart from a linear transformation. As discussed already in chapter 3.6, straight tracks transform into straight tracks
in this projection, as $\frac{Y}{X} = a + b\frac{1}{X}$. Therefore the arms of the V are straight, their gradient $b$ is a measure of the particle momentum and the V-direction gives its charge, so that one gets the same features as for the V-Plot of the ALEPH TPC. One feature is especially important, namely that both, momentum and charge, can be estimated locally from the display of the TPC hits only without displaying the vertex point.

Figure 20 shows a simulation of track hits with either 2, 6 or 12 GeV/c in $Y/Z$ and as V-Plot. It is evident, that the shape of the track images, the V’s, depends only on momentum and not on the position of the tracks.

The NA35 event shown before in figure 19 is displayed in figure 21 as a V-Plot, on which the tracks are easily identified, and from which one can extract momentum and charge of the particles.

### 6.8 Interactive Correlation of Different Windows

It is often convenient to work on several windows synchronously

- with the same projection but different magnifications,
- with different projections,
- with the same projection and magnification but different data.

The following problems may arise:

- Identify the same object on the same window, if the object is large or is drawn several times.
- Identify the same object on different windows.
- Identify the same 3D position in different windows.

Correlation of objects may be done either through

- colors (chapter 2.4) or
- "pick and move"
- "pick and blinking" or highlighting.

The pick and move method works as follows:

1) an object is identified by picking on a selected window.
2) the pointer is moved interactively to a different position or window.
3) the pointer is moved by program to the closest occurrence of the previously identified object.

Instead of moving the pointer to the previously identified object, the object itself may be blinked or highlighted on all projections.

To identify a position on different windows with different projections one calculates the projections of an artificial 3D point on all windows and draws corresponding pointers at these positions. Via the mouse a selected pointer can be moved, so that the coordinates of the virtual point are modified. The pointer positions in the other views move accordingly. This method is most effective, if one of the projections is the V-plot, as most tracks can be identified through their direction.

---

15 The data originate from one of the very first events ever recorded in the NA35 TPC, which was not yet well aligned.
16 This method is not applied in DALI.
The Puzzle-Plot, a Three Dimensional Representation of Calorimeter Data

Figures 16d, e, f show three layers of a calorimeter side by side, so that one can follow the shower development by comparing the three pictures, as discussed before. It is possible to recognize, which cells in the different layers form a cluster, i.e. belong to the same shower. Thereby it is possible to verify independently the clustering algorithm of the pattern recognition program. With increasing number of layers, however, this gets more and more tedious.

An example of a calorimeter with a large number of layers is the ALEPH Silicon Calorimeter, SICAL. It has a cylindrical structure and consists of two parts 2.5 m down and upstream from the interaction point with a length of 12 cm, an inner radius of 6 cm and an outer one of 14.5 cm. Each cylinder is divided into 12 disks, each disk is divided into 16 rings and each ring into 32 angular sections with $\Delta \varphi = \frac{360^\circ}{32} = 11.25^\circ$. Thus the detector consists of 6144 cells in 12 axial and 16 radial layers. Each of the disks is rotated by $\Delta \varphi/3$ with respect to the previous one, i.e. the disks are “staggered”.

In color plate 12a all active cells are displayed in the wire frame technique in a perspective view. A section is blown up in color plate 12b. All blue cells are considered by the clustering algorithm to belong to the same cluster. The same is true for the pink cells. The yellow cells do not touch these two clusters and have too little energy to form a cluster by themselves, i.e. they are regarded as noise by the clustering algorithm.

However, the picture is very complicated due to the large number of active cells and due to the fact, that perspective drawings are less well comprehensible if lines are curved and the objects have different orientations. The picture would get even more complicated if in addition one would try to display the amount of energy deposited in the active cells.

By use of such pictures it is possible to prove, that clusters do not touch each other, if at least one suitable viewing direction can be found, where they appear separated (see the blue and pink clusters in color plate 12). It is more tedious to prove that all noise cells (yellow) are not connected to one of the clusters, as it may be necessary for this purpose to select a different direction of view in each case. For complex clusters, however, it is impossible to identify visually all cells belonging together. The Puzzle-Plot, explained below, offers a way out. However, it is not intuitively understandable.

The rules to interpret the Puzzle Plot are best explained by ignoring for a moment, what is known about the calorimeter. Color plate 13a shows a Puzzle Plot and color plate 13b the blow-up of a section. The blow-up shows 10 black fields separated by white lines, where each field contains one or several “triple crosses”. A triple cross is composed of one vertical bar and three equidistant horizontal bars of identical color. The yellow one in color plate 13b may serve as an example. The crossing points of the bars lie along three diagonal black lines and are emphasized by a black spacing between the bars. These lines become clearly visible if there is more than one triple cross in one field.

Next we define the connection between all connected triple crosses applying the following rules:

Triple crosses in neighbouring fields are connected if:
1) one (or two) of their horizontal bars touch each other, or
2) their vertical bars touch each other.

Triple crosses in the same field are connected, if
3) their vertical and horizontal bars have minimum distance between each other, which in this example is equal to the distance of the pink bars in one field.
Triple crosses in non neighbouring fields are connected, if there is a path of connected neighbouring triple crosses between them, applying the rules given above. One sees clearly, that all blue triple-crosses in color plate 13b are connected directly or indirectly via other triple crosses. The same is true for the pink ones, whereas the yellow one is unconnected.

There is a close correspondence between these rules and the problem to find clusters in the SICAL, as the black fields are a mapping of the towers arranged vertical to the Z-axis and the triple crosses describe the position of the active calorimeter cells. The position of a triple-cross is defined by three independent variables. Two \((H,V)\) define the position of the field in which it is drawn, and one \((D)\) defines the position of the triple-cross inside the field along the diagonal lines. The position of a cell in the SICAL is defined by \(\phi, Z, \rho\). If one sets \(V = \phi, H = Z\) and \(D = \rho\), the calorimeter is mapped onto the Puzzle-Plot, so that it is possible to visualize the cells in all three dimensions. This allows to check the association of cells to clusters.

The arrangements of the fields reflects the staggering, which necessitates the use of three horizontal bars in order to visualize the connection in Z-direction (rule 1 above).

One notices from color plate 13a, that in the fields denoted by a `*`, triple crosses exist, which cannot be connected to the blue cluster, directly or indirectly. This is due to the fact, that the real clustering algorithm of the ALEPH SICAL allows the connection of cells, if they touch at a border lines parallel to the Z-axis, i.e. triple crosses in neighbouring fields above or below are connected if:

4) their vertical lines are displaced by one step.

The application of this rule is easy in the puzzle plot, but very difficult in the wire frame representation.

The special form of the Puzzle-Plot as described above may of course vary with the structure of the calorimeter displayed. The triple-crosses can be regarded as symbols displaying the depth of the cell. There may be other symbols useful for this purpose.

The representation of the SICAL data via the Puzzle-Plot allows in a simple way to represent in addition the energy deposit in the active cells. At the center of the three horizontal lines of a triple-cross a horizontal line of a different color (red and white in our example) is drawn, the length of which is proportional to the deposited energy. Thus one gets the visualization of a three dimensional scalar field.

### 8 Classification of Pictures

The pictures discussed in this paper may be grouped into three classes:

1) realistic pictures:

They are obtained by either a sequence of rotation, linear scaling and projection or by central perspective, in both cases cutting away parts of the detector, if useful. The resulting pictures are intuitively understood.
2) schematic projections
They comprise

• all other linear transformations, e.g. a change of aspect ratio.
• the Y/X projection with fish-eye.
• the $\rho'/Z$ projection with or without fish-eye.

In many cases schematic projections are better understood and easier to use than the realistic projections. The best known projection of this type from another field is the Mercator projection used in geography.

3) abstract pictures:
They show no resemblance to the original objects. They comprise the angular projections $\phi/\rho$, $\phi/z$ and $\phi/\tau$ as well as the V-plot and the puzzle plot. They are not intuitively understandable and need training to be understood. Therefore, they are not suited for general presentations but they are usually very powerful.

Often only realistic pictures are implemented on high end graphics systems, such reducing their capabilities.

9 Conclusions

Visual representations are used for two different purposes, namely for

• presentations, i.e. talks and papers,
• visual analysis.

If pictures are used in presentations they should be intuitively understandable, without requiring long explanations. If a picture is not just used to catch the eye, the information the lecturer or writer wants to pass on to his audience must be comprehensible from the picture. This is even more demanding, if a picture is shown in a talk for a short time only. To this aim pictures must be sufficiently simple, still matching the complexity of detectors and events.

In the case of a cylindrical detector the best pictures are obtained using cross-sections, i.e.:

• $\rho'/Z$ as side view and $Y/X$ as front view, with the endcaps omitted in $Y/X$. If these projections are applied, the various subdetectors do not overlap, so that both hits and subdetectors can be drawn together and hits fall onto the image of the subdetector, by which they were recorded.

These images can be further improved by:

• applying (non linear) fish eye transformations, so that the inner chambers are enlarged and the outer ones reduced in size;
• displaying the energies deposited in the calorimeters as histograms in the form of structured areas;
• coloring the subdetectors;
• choosing suitable colors for hits and tracks. Tracks are better separated, if neighbouring ones are colored differently.

Three dimensional information can be transmitted by

• showing $Y/X$ and $\rho'/Z$ side by side. In this case color may be used to correlate objects in the different pictures.
These methods yield clear pictures in the case of experimental setups like the ALEPH detector. Furthermore, it is of big help to carefully select events, which show the required features, but which also give good pictures.

However, if events with very high multiplicities are to be shown, a display of the data may yield useless pictures, i.e. the limits of these methods may be met. A way out might be the application of one of the following methods, namely

- to draw the data in a simpler form, i.e. tracks (lines) instead of hits (points),
- to restrict the amount of displayed data by use of information given by the pattern recognition program, e.g. by a cut on track momentum,
- to display data from sufficiently small volumes, where the problem of how to find and define such a volume arises,
- to use non-conventional projections like the V-Plot or the Puzzle Plot, which yield clearer pictures. However, they have the big disadvantage, that the listener or the reader is required to have the necessary knowledge for their interpretation.

The above conclusions are also valid for the visual analysis of tracks, where it is often necessary to display the basic data, i.e. display the hits instead of tracks. It is normally not possible to restrict the analysis to specially selected, clear events.

Most events have a large amount of tracking information. Track recognition is drastically improved through the following concepts:

- Track compression, i.e. a low magnification in track direction and a large one perpendicular to it.
- Track Linearization, which facilitates compression, but also recognition and visual extrapolation of the tracks. In the case of helices this is achieved using angular projections.
- Use of projections which allow the local estimation of track features, e.g. particle charge and momentum.

Through the first two methods it is possible to increase the magnification of a track in its full length to a level, where the errors of data recording become apparent, i.e. the limits imposed by screen and eye resolution are overcome.

If three dimensional data are available, two further concepts get important, namely

- the use of orthogonal projections,
- the overlay of two projections to transfer 3D information.

A good realization of the above concepts is found in the V-Plot, the mathematical formulation of which depends on the experimental setup. Furthermore, the V-Plot is a powerful means to extrapolate tracks to calorimeter representations like the Lego Plot.

Whereas the limits of the conventional methods seem to be reached when examining difficult events, the limits of the V-Plot technique still seem to be further away. This technique is however constrained to real three dimensional data, i.e. 3D hits and 3D track segments.

Calorimeter data are best displayed as Lego Plot or through pictures with similar structure, e.g. $\phi/\theta$, but different energy representation. Different layers projected side by side allow to judge the shower development, where the island representation helps to associate the clusters in different layers.

The use of this technique, however, gets more and more tedious with increasing granularity of the calorimeter, i.e. if many layers exist. A way out is shown in the Puzzle Plot, although it may be limited due to screen and eye resolution.
In short, the aim of visual representations, namely to transfer data from the computer into the human brain, can still be accomplished in a fast, unambiguous and efficient way. Even for more complex detectors and events, display methods are available to present the full data. However, the price to be paid is the use of more abstract representations. If one is ready to accept this complication, visualization of events will continue to serve as a helpful tool for representation and analysis.

10 Acknowledgements

The successful construction and running of the ALEPH detector by the collaboration provided the events, for which the methods described in this paper were developed. During development of these techniques we have had input from many of our colleagues in ALEPH. We owe special thanks to Chris Grab, Salvador Orteu, Mark Parsons, Raimund Vogl, Gabi Waltermann and Rongfen Xu for their valuable contributions to the program and to Brigitte Bloch-Devaux, Jürgen Knobloch, Dieter Schlatter and Jack Steinberger for helpful discussions and suggestions. For information on NA35 we thank Ingo Schneider and the NA35 collaboration for providing us with data. For help with the program FrameMaker we acknowledge gratefully the help of Herbert Mettler and Mario Ruggier.

11 References

[4] This method was proposed by M.Mermikides.

12 Further Reading

The RD13 DAQ System and the Object Management Workbench

Bob Jones
CERN, Geneva, Switzerland

Abstract

This paper describes the progression towards an object-oriented (OO) environment for software development as used in the RD13 research project for data acquisition systems of high energy physics experiments. A general description of the RD13 DAQ system and the existing software development environment is followed by an overview of the selected OO method and associated CASE tool. How the method and CASE tool relate to the existing environment is explored. As examples, a number of applications developed using the OO environment are explained. The paper concludes with an attempt to indicate what will be the most significant changes for DAQ software development between the present day and start-up date of the LHC detectors.

1 Introduction

The aim of this paper is to describe the transition from a structured analysis and design based software development environment for DAQ systems towards one based on object-orientation. We start with a general description of the RD13 DAQ system, its motivations and goals and the software environment (i.e. based on structured analysis and design principles) used to develop the current version of the software. The tools and techniques in this environment are described from the developer’s point of view. We then give an overview of the selected OO method and its associated CASE tool. The relationship between the OO techniques and tools to the existing environment is explored. A number of DAQ components have been built in this OO environment and their implementation is outlined. The reader should bear in mind that many OO methods and tools exist today and the examples provided are not the only choices available but those most suited to the RD13 project at the time of evaluation. The tour concludes with an attempt to indicate what will be the most significant changes for DAQ software development between the present day and start-up date of the LHC detectors.

2 Overview of the RD13 DAQ system

The RD13 project was approved in April 1991 for the development of a scalable data taking system suitable to host various LHC studies [12]. The basic motivations come from the conviction that, being too early for a 'top-down' design of a full DAQ architecture, a lot can be gained from the study of elements of a read-out, triggering and data acquisition and their integration into a fully functional system.

2.1 Main goals

The time scale for LHC experimentation and the inadequacy of the existing read-out and DAQ technology make a 'top-down' design premature. A more appropriate preparation for LHC is to spend some time and resources in investigating system components and system integration aspects. The investigation is more effective if done in a realistic environment, such as the data taking phase of a detector prototype at a test beam. Such a setup has the double advantage of serving the increasing data taking demands from the evolution of the detector read-out and
triggering electronics and helping the smooth evolution of the DAQ system by forcing a continuous integration of newly developed components into a working environment.

A further motivation drives RD13, the conviction that the traditional standard High Energy Physics methods for online software developments would fail, with the obvious disastrous consequences for LHC experimentation, given the undeniable complexity of the required systems. Much has to be done to find suitable methods for complex online system designs and much has to be learned in the area of software engineering tools. The ability to be constantly evaluating and experiencing in an environment close to the real one, is the great advantage of a 'down-to-the-earth' learning ground, as proposed by the RD13 collaboration.

2.2 Prototype DAQ

A test beam read-out facility has been built for the simultaneous test of LHC detectors, trigger and read-out electronics, together with the development of the supporting architecture in a multiprocessor environment. The aim of the project is to build a system which incorporates all the functionality of a complete read-out chain. Emphasis is put on a highly modular design, such that new hardware and software developments can be conveniently introduced. Exploiting this modularity, the set-up will evolve driven by progress in technologies and new software developments.

![Diagram](Image)

**Figure 1**: DAQ Hardware setup at ATLAS test-beam 1994
One of the main thrusts of the project is modelling and integration of different read-out architectures to provide a valuable training ground for new techniques. To address these aspects in a realistic manner, the group collaborates with detector R+D projects in order to test higher level trigger systems, event building and high rate data transfers, once the techniques involved are sufficiently mature to be tested in data taking conditions. The complexity expected for the software online system of LHC experiments imposes the use of non-traditional (within HEP) software development techniques. Therefore, the problems of data acquisition and support software are addressed by the exploitation of modern software engineering techniques and tools.

2.3 Test-beam activity

The RD13 DAQ system has been in use since November 1992 by ATLAS detector prototypes for their runs in the H8 area. The setups have included different configurations both in single and multi-detector mode including the hadron calorimeter [16], Liquid Argon calorimeter, transition radiation tracker [15], silicon detector [17] and second level trigger prototype [18].

3 Existing Software Development Environment

The current version of the RD13 DAQ is developed according to the principles and techniques commonly known as SA/SD (Structured Analysis/Structured Design)[13] [14]. As a basis for the development we chose UNIX, as the common operating system, C as the programming language, NFS (Network File System) as the means of file sharing and TCP/IP as the common protocol for communication. The major elements of any application are its access to data and parameters (i.e. databases), its functional part (i.e. the actions it is to perform) and its interface with the user (i.e graphical user interfaces) and other applications (i.e. distributed computing.) For most of these elements we have adopted a technique and associated CASE tool which are now described.

3.1 DataBases

A data acquisition system needs a large number of parameters to describe its hardware and software components. In short, four distinct databases can be envisaged for a DAQ to store such information: Hardware configuration (describes the layout of the hardware in terms of crates, modules, processors and interconnects); Software configuration (describes the layout of the software in terms of processes, services provided, connections and host machines); Run parameters (e.g. run number, recording device, Level 2 trigger state etc.); Detector parameters (information pertaining to the detector and defined by the detector group themselves within a fixed framework).

3.1.1 Data Access Library (DAL)

The database is accessed by DAQ programs via a data access library. A DAL provides an interface to the queries and updates needed by the applications, that is the DAL implements only those database operations needed (and allowed) on the data at run time. The DAL hides the actual implementation of the database and the data model to the application. This means that the underlying database system can be replaced without affecting the application programs.
3.1.2 Implementation: QUID

An alternative to commercial relational DBMS are in-memory bank systems. They do not have all the functionality of the relational DBMS but also do not have so many overheads in terms of performance and demands on the underlying operating system. One such commercial in-memory system is QUID from Artis srl [10].

QUID is a database development environment targeted to real time applications (i.e. where performance is needed and the full functionality of a DBMS is not) consisting of the following components:

- An Entity-Relationship (E-R) diagram graphical editor, implementing an extended E-R model.
- A query language: it extends the C language by providing statements to manipulate entities and relations and to navigate the database. Actions, expressed in C code, to be performed on the data can be attached to the QUID statements.
- A code generator: this takes as input both a database definition (to generate all the code needed to manage the database) and a QUID program (to generate a C module operating on the database.)
- User interface (modifying and browsing a database): a tool which, given a database description as produced by the QUID editor, can produce an application to navigate and modify the contents of a database. This is an important point, since it provides a way to interactively access the data without the need for DAQ developers of build dedicated graphical frontends.

![QUID Diagram](image)

Figure 2: QUID editor and browser showing one view of the software configuration database

The database is kept in the application’s memory and its contents can be saved/loaded to/from a disk file. Multiple versions of a database (e.g. different system configurations) may be maintained by saving/loading to/from different files. There is no provision for distributed transactions, nor for the distribution of data among many processes. For read-only databases
this restriction is easily overcome by having all participating processes access the disk files via NFS. When writing is also needed, a special scheme has to be devised.

3.2 Distributed computing

Sophisticated distributed DAQ systems, such as the RD13 DAQ, demand specialised software in order to control all the aspects of the data acquisition system. Processes need to cooperate to perform processing functions, and to share data and information. It must be possible to synchronise processes and monitor their progress. UNIX does not provide all the services and facilities required to implement such applications and so we have looked elsewhere to find tools that can provide the missing features.

One such tool is ISIS [2], a commercial toolkit for distributed and fault-tolerant programming. The toolkit is a set of fault-tolerant software protocols that are accessed by an application programmer interface. Included is support for groups of cooperating processes, replicated data, distributed computation and fault tolerance.

Applications that use ISIS are organized into process groups. The membership of a process group can change dynamically, as new processes join the group or as existing members leave, either by choice or because of a failure of some part of the system. A process can be a member of many process groups. Messages can be sent, or broadcast, from a process to a process group so that all the members receive a copy, without explicitly addressing the individual members. A process broadcasting a message can indicate that it wants to wait for replies from the recipients of the message. Process groups provide a convenient way of giving an abstract name to the service implemented by the membership of a group. ISIS has been used as the basis for communication in the components of the DAQ including the run control system.

3.3 User interfaces

Today, most computer systems use some graphical display to communicate with the user. The X Window System [1] has become a de-facto standard package for building user-interfaces on bit-mapped computer screens. The X Window System offers a relatively low-level set of functionality and so toolkits have been built on top of it to simplify the task for the Graphical User Interface (GUI) developer and to ensure a uniform interface across applications. The X Window System based GUI toolkit that is available on the largest number of computer systems is Motif [1] from the Open Software Foundation (OSF)[1]. The RD13 project adopted Motif to run on its network of heterogeneous UNIX workstations. The Motif toolkit offers a higher level of abstraction to the GUI developer than the XWindow libraries and so simplifies the programming task. Nevertheless, programming an interface with Motif remains a complicated task and so we have adopted a GUI builder tool, called X-Designer [3], which can automate some of the code production.

Using X-Designer, the programmer can interactively design the user interface of an application by selecting the appropriate Motif widgets from a palette. It also offers the possibility of incorporating 3rd party widgets (such as the XRT 2-D graph and table widgets [1]) and allowing them to be used alongside the standard Motif widget set. When the interface satisfies the developer, he can ask X-Designer to generate C code with embedded Motif calls needed to build the interface. This code can then be integrated with the application to produce an executable program. We have developed interfaces to a number of components of the DAQ with the help of this tool.
3.4 Online Help for Users

It is important for users of the DAQ system to have access to the relevant documentation online. Such documentation describes how the various components of the DAQ work, how they interface with each other and how the user can control and configure them. We use FrameMaker [2] to write such documents then convert them to HTML (WWW's mark-up language) using an automatic converter program (i.e. WebMaker developed at CERN). The documents are viewed online using the Mosaic browser which is controlled by the DAQ user-interfaces via a small library we have developed based on the Mosaic application programming interface.

3.5 Software organization and distribution

In a software development of any size, especially when more than one developer is involved, it is important to have a disciplined approach to the organization of the various components and modules.

3.5.1 Directory tree

A directory tree of the code sources, include files and binaries for a version of the DAQ software provides a basis for a logical subdivision of the components (or products). A version of the DAQ includes: DAQ components (e.g. the run control, the data flow modules), libraries (e.g. database access library, vme bus access library) and facilities (e.g. error messages).

This common approach to storing sources and binaries means there is a unique place where a version of application programs, libraries and include files can be found without (or
with minimal) interference to the developer's work.

A standard account contains the sources for the current production version of the software and a makefile to rebuild, from those sources, the product (even in the absence of the author). The account is managed by one person, to whom a new version of a software product (for production or validation) will be given. A single access point to the modification of the contents of the standard directory tree guarantees proper notification of new versions and avoids the temptation of doing a “quick fix” to a module.

![Figure 4: DAQ software directory tree](image)

3.5.2 Code repository

The management of source code, organized in the above directory tree, to track software modifications, releases and configurations is an important activity in the software project. The source code of the different products is organized in a repository managed by CVS [9]. Each product is maintained in several releases and the full data acquisition system releases are superimposed on to the structure.

3.5.3 DAQ builder

The DAQ builder is a shell script and a makefile developed for the management of the DAQ software components. Both the needs of the integrator, who is responsible for building a whole DAQ system, and the individual developer, who is responsible of the individual products, are taken into account. The tool is based on the directory organization and code repository described above.

4 Object-Oriented method and CASE tool

One of the aims of the RD13 project was to see if new techniques, such as Object Orientation, could be applied to DAQ systems. The existing software development environment covers a large proportion of the software that needs to be implemented for the DAQ system. However, there are still some areas (such as the functional code of applications) for which there is no direct support from the set of tools and other areas are not supported over the whole life-cycle of the software. Also, the integration of the software generated from by the various tools is not always obvious. To improve this situation, we decided to evaluate an object-oriented method and CASE tool.
4.1 Overview of Object Oriented Information Engineering (OOIE) Method

The Object Oriented Information Engineering (OOIE) Method [4] is closely related to the Ptech method originating from Associative Design Technology in the US.

The method makes a clear division between object structure (static) and object behaviour (dynamic). Analysing the object structure produces a set of diagrams showing object types and their associations, inheritance and aggregation. The object model is quite rich with support for multiple and dynamic classification. An analysis of the object's behaviour produces a set of diagrams defining sets of operations that may occur in the system in terms of the operations themselves, event types, trigger rules and control conditions. In the latest revision of the book the authors class the event based definition of object behaviour as scenario-related and offer an alternative state-related definition based on finite-state machines (FSMs).

4.2 Overview Object Management Workbench (OMW) CASE tool

Object Management Workbench [6] is a commercial CASE tool sold by Intelllicorp. The CASE tool supports the OOIE analysis and design method described above over the full life cycle from analysis through to code generation and testing. The tool set includes:

- **Object Diagrammer**
  an interactive diagramming tool to define objects of the application model, their relationships, attributes and composition.

- **Event Diagrammer**
  an interactive diagramming tool to define the behaviour of the model in terms of events and operations triggered by events. Once defined, the behaviour can be simulated before the code generation stage.

- **Business Rules Editor**
  a syntax directed editor for defining rules that constrain the behaviour of the model by adding conditions to the execution of operations.

- **Interface Workbench**
  a direct manipulation tool for WYSIWYG creation of graphical user interfaces. All screens created with the Interface Workbench are fully portable across UNIX workstations running the X Window System and Microsoft Windows-based PCs.

- **Data Linkage Editor**
  creates and manages connections between graphical components and objects defined in the application.

- **Scenario Manager**
  a testing and simulation tool used to execute and validate the models created in OMW. Scenarios can be stored on disk and reloaded at a later date to test new versions of the models.

- **Report Browser**
  allows the user to automatically generate documentation and check problems on the developing application.

The above tools are built on top of the Kappa [7] programming environment. The Kappa system is an ANSI C-based visual environment for developing and delivering distributed applications in open systems environments. The Kappa system architecture is modular and layered so that applications contain only those portions of the system needed at execution time, resulting in smaller applications with better performance. Kappa is based on a core engine with a complete ANSI C programmer's interface (API). The system includes the following tools:
• ProTalk
  a high-level language of the Kappa System. It is a hybrid of a 4GL and OO language that
  gets compiled into ANSI C. Alternatively the developer can use ANSI C directly.
• ProTalk Workbench
  an interactive symbolic debugging tool for developing with ProTalk (both interpreted or
  compiled into ANSI C).
• C Listener
  provides a C interpreter, symbolic debugger and various other programming environment
  tools
• Probes
  customizable interface components used for simplifying and customizing the development
  environment.
• Object Browser
  provides inspection, creation, and manipulation of objects and their attributes, and manages
  the organization of programs and the creation of run-times.
• Active Relations
  a simple spreadsheet-like mechanism for expressing integrity constraints, derived values,
  and complex interrelations.
• Monitors
  are special-purpose Kappa objects that are attached to attributes of other objects and execute
  some behaviour when the attribute is accessed or modified.
• Integrated Data Access
  Kappa provides bidirectional transfer of data between relational databases and Kappa
  objects by dynamically generating SQL to map records and tables into Kappa objects.
• Integrating Legacy and Third-Party Systems
  since Kappa is ANSI C based, it is possible to integrate external code and libraries into
  Kappa applications. Kappa applications may be fully embedded and support standard ANSI
  C linking conventions. Kappa’s run-time function library permits it be called or embedded
  within any C program.
• Distributed Objects
  Kappa applications can be distributed between UNIX workstations and PCs running Win-
  dows by using the Kappa CommManager. The CommManager provides transparent com-
  munication among distributed objects by running over TCP/IP networks and complies to the
  CORBA draft protocols defined by the Object Management Group (OMG).
Applications developed with OMW/Kappa can be divided into domains or subject areas of an
application in which all object references are unique. The OMW/Kappa development tools run
on Sun (SunOS and Solaris) and HP workstations. The developed applications can also be
ported to IBM compatible PCs running Microsoft Windows.

5 Applications developed using OMW

We have used the OOIE method and OMW CASE tool to develop a number of applications
which have become integrated components of the RD13 DAQ.

5.1 Error Message Facility

We have used OMW to develop a replacement for the existing tools for producing and
maintaining unique error codes within the DAQ software. The original version uses the
GENMESS facility (a porting of the VMS MESSAGE facility to UNIX) to define unique error
messages but it has several limitations including no automatic means of avoiding collisions between error codes defined in separate files.

![Diagram of EMF Instrument Workspace](image)

**Figure 5:** OMW Object Diagram for Emf error message database

The GENMESS error code definition files have been replaced with a database of error messages implemented using OMW's object repository. Two application programs work on the database - EmfEditor and EmfTrans. EmfEditor is used by the developers off-line to interactively modify the definitions of error messages used by the RD13 DAQ system. The error codes are loaded from disk, manipulated using the graphical user interface and then saved back to disk. A C include file containing all the error code definitions can be generated for use by applications from the graphical user interface. The definitions are used online by the EmfTrans application.

EmfTrans is an online server process that performs the translation from error code to textual error message. Application programs exchange messages with EmfTrans via ISIS just before they report error messages or when they simply want to retrieve the text associated with error codes. EmfTrans loads the database of error message definitions and connects to the distributed run control system. When it receives a translation request from DAQ processes it looks up the given error code in the database and returns the associated textual message.

### 5.1.1 Design and implementation

The EmfEditor and EmfTrans applications are composed of several domains:

- **EmfDM** data model that defines the classes used to describe error message definitions. This domain consists of object structure diagrams (see Figure 5.)

- **EmfDMI** holds the instances of the classes defined in EmfDM. EmfDMI represents the "database" of EmfDB and is loaded/saved to disk by EmfEditor and loaded (but not saved) by EmfTrans. No diagrams or code are held in this domain only object instances.
Figure 6: EmfEditor main window

Figure 7: OMW main event diagram for EmfEditor
- EmfIO defines the methods used to save/load EmfDMi to disk, scan GENMESS files (for migration purposes) and generate the C include file for error definitions. The domain contains event diagrams and methods written by hand in C and Protalk.

- EmfEditor defines the graphical user interface used to modify the contents of EmfDMi. Screens are defined using the interface workbench and the data Linkage Editor is used to link them to object instances from the EmfDMi domain. The on-line help is provided by linking to the same Mosaic browser library as in the existing development environment.

- EmfTrans interfaces with ISIS and the run control system so that, when run as a process, ISIS requests for error code translation are received, queried in the EmfDMi “database” and replies sent. Event diagrams, methods and a module of hand-written code are included.

### 5.1.2 Searching the database

The database queries are implemented as methods in the ProTalk 4GL OO language. ProTalk has the notion of knowledge expressions for accessing information about objects in domains which can be used with backward chaining to find solutions to statements. For example, the following method retrieves an error definition object given the value of it’s message text attribute.

```plaintext
method findMsgByMsgText.Run! (?value)
{
    find1 ?ErrMsgDef.msgText == ?value;
    return ?ErrMsgDef;
}
```

**Figure 8:** ProTalk method for retrieving an object given the value of an attribute

### 5.1.3 Application distribution and archiving

The above applications are integrated with the DAQ software distribution scheme as a single DAQ component. A single top-level makefile was written by hand to start the more detailed makefiles generated for each domain by OMW. The source code files for the object methods and the files that define the model for the CASE tool are archived under CVS and integrated with the DAQ builder set of scripts.

### 5.2 Online Volume Bookkeeping

The Online Volume Bookkeeping (OVBK) system is designed to provide an automatic log of the data recorded by the DAQ system. The OVBK retrieves information from different DAQ modules and databases to complete the log. The data describing the run configuration, run date, archive file information and run quality are maintained in a database, which is implemented using OMW’s object repository. The OVBK system also provides a graphical user interface to interactively access the database and to generate reports. This application required integration with the run control system, the run parameters database and the error message facility. It is implemented as a process on a workstation while the DAQ is taking data and writes the details of the current run to the database. The graphical user interface is run as a separate process offline. The intention is to use the Kappa CommManager as a means of performing transparent inter-process communication between the graphical user interface and the OVBK server to allow the user to modify the contents of the database on-line.

The OVBK is implemented as several domains in a similar fashion to the EmfEditor and EmfTrans applications. It is also integrated with the DAQ software distribution and archiving.
scheme.

![OMW Object Diagram for Online Volume Bookkeeping database](image)

Figure 9: OMW Object Diagram for Online Volume Bookkeeping database

Much use is made of trigger rules to decide what action should be performed according to the messages received from the DAQ system.

![Example trigger rule built using the OMW Rule Editor](image)

Figure 10: Example trigger rule built using the OMW Rule Editor

5.3 Hardware Database Editor

The hwdbEditor program allows the user to edit the contents of the hardware database used by the RD13 DAQ system. The user can modify the existing configuration, define new hardware modules or delete obsolete ones. The aim of this application is to replace the standard QUID editor which requires knowledge of how the database was implemented in terms of the entity-relationship (E-R) schema.

The hwdbEditor application uses the Object Diagrammer to make an OO equivalent of the QUID E-R model. The actual contents of the hardware database are loaded at run-time from QUID using the data access library and instances of the object types defined on the Object Diagram are created. The user can then manipulate these instances via the GUI. Finally the modified contents are written back to QUID onto disk via the access library. The editor is implemented as several domains in a similar fashion to the EmfEditor application. It is also integrated with the DAQ software distribution and archiving scheme.
5.4 OMW/Itasca interface

QUID has been successfully used to implement all the four DAQ databases mentioned earlier. This situation is satisfactory for the present but with the arrival of Object-oriented DataBase Management System (ODBMS), it may be possible to extend the functionality of the databases. We have made a prototype implementation of the DAQ databases using a commercial ODBMS called Itasca [8] and, from our initial impressions, it appears that ODBMS have a lot to offer the online environment. Not only can ODBMS be used to implement existing database functionality but also ODBMS promise to be better placed to address the issues of unstructured data, space requirements and performance which had imposed the development of ad-hoc solutions. Many ODBMS are capable of handling large unstructured data (e.g. images), provide transparency in case the database is physically on more than one device and support object caching and clustering.

As an evaluation, we have developed a prototype application that provides a link between the OMW CASE tool and the Itasca ODBMS. The motivation for this prototype was to see if an alternative technique for object persistency could be used with OMW and also to provide a graphical schema editor for Itasca. The prototype is implemented as an OMW application that uses the Kappa library to access meta-data about object diagrams and Itasca’s C client library to access corresponding definitions in the database. The prototype can define classes with attributes and relationships including inheritance and cardinality constraints, store OMW object instances in Itasca and reload instances from Itasca to OMW. The prototype highlighted a number of differences between the respective object models of the Kappa environment and the Itasca database. A limitation with such a technique is that the OMW object methods could not be easily stored in the Itasca server because they are not written in LISP. But there is one important advantage of using Itasca over OMW/kappa internal object persistence - the objects from Itasca can be saved and restored on demand. With Kappa’s object persistence all the ob-
jects of a domain are either stored or retrieved on block. This means that to save one object you must save all objects in the domain and that the size of the object base is limited to that of the virtual memory allocated to the process. However, we have shown that it is possible to provide such a link between the CASE tool and the database and that such a technique could possibly be used as a means of CASE tool independence or migration (e.g. store applications developed with OMW in Itasca then read it back out by another CASE tool).

![Figure 12: OMW event diagram of OMW to Itasca translator](image)

![Figure 13: ITASCA Active Data Editor: View Graph of EmfDB Kappa objects stored in Itasca](image)

6 **Comparison of OMW/Kappa with existing development environment**

If the OMW/Kappa applications described above were built using the older (i.e. SA/SD) set of development tools then the architecture would probably look like this:

- Use QUID to define an entity-relationship diagram representing the structure of the data stored using the OMW/Kappa object repository. Note that the notion of inheritance would not be available and hence a more complicated schema would need to be devised for the applications such as the hardware database editor and online bookkeeping system.

- Use the QUID query language to define routines for a data access library to create, modify, query and delete entities and relationships defined on the E-R diagrams. In OMW/Kappa such constructs are already defined in the ProTalk language and the knowledge expressions can be used for complex queries.

- Use X-Designer to develop the graphical user interfaces. The routines and callbacks to display the entities in the QUID structures on the user interface, modify their values and validate the user input would need to be coded by hand. In OMW these interfaces are
implemented using the Interface Workbench and the Data Linkage Editor provides the relationship between domain objects and their representation on the screen. User input can be validated using Kappa Value Checks.

- Use ISIS for inter-process communication. Map the required communication onto process groups and a number of message types and write the code to send/receive the information. We only have limited experience with the Kappa CommManager but it seems very easy to use and has little impact on the development of the application. Using the CommManager we simply make remote method executions as if the object were defined locally (i.e. OO RPCs).

- Code to guide the execution of the applications would need to be developed by hand (i.e. the equivalent of the knowledge that is stored in the OMW event diagrams).

- Makefiles would need to be written by hand where as they are generated automatically by OMW.

### 6.1 Assessment of OOIE Method

We have found the OOIE method to be clear yet sophisticated with sufficient support for static object definition. The objects structure offers such concepts as specialization, generalization, classification, composition and partition. There are constructs in the event diagrams to represent synchronization and concurrency which would help with distributed applications but it lacks other concepts for real-time work (e.g. priority, exceptions etc).

Through our prototype interface between OMW and Itasca, we found that the OOIE object model as supported by OMW and Kappa is very rich and that many features could not be implemented in Itasca (e.g. user-definable Object Identifiers, monitors and slot formulae).

### 6.2 Assessment of OMW CASE tool

There are several facilities in the method which would be useful to us but are not supported by the CASE tool:

- timer events
- finite-state diagrams as an alternative means of representing object behaviour (as opposed to event diagrams)

Some other features of the method are only partially supported by the CASE tool:

- event diagrams are not re-entrant which means they cannot be used recursively
- the code generated for event diagrams is single threaded and so concurrent operations are performed sequentially and the order cannot be controlled.

#### 6.2.1 Incremental development

The incremental development cycle supported by OMW is very practical and one of the best features of the tool. One of the most important advantages of using OMW is the clarity of the diagrams produced (i.e. event and object diagrams). Once a developer knows the notation (which is very simple when compared to other notations such as Booch [11]) he can quickly get an impression of how an application is structured even if it was developed by someone else. The diagrams are guaranteed to be up to date and complete since they are used to generate the runtime code.

The principle of object domains allows the developer to modularise the application into
groups of closely related objects. OMW is a very open tool as shown by the number of third party software packages that we have been able to integrate with our applications.

The built-in simulator allows applications to be tested before generating a run-time. This means the developer can remove the most obvious bugs without leaving the tool. It is also an aid to other developers who need to perform maintenance on the software by helping them understand how it works via the animated event diagrams. In general we have found that those bugs which persist beyond this stage of testing are usually related to integration with third party software (e.g. ISIS or other DAQ components) and effects of speed and space differences between interpreted and compiled code (the simulator interprets the application code while at runtime it is compiled).

The report generator is helpful for debugging purposes since reports can be made to list problems found by the tool on the various diagrams of an application.

6.2.2 Programming environment

The underlying Kappa programming environment is very rich and sophisticated. We have not used all of the features provided but have found the built-in container classes and object manipulation facilities very useful. The ability to modify the object structure dynamically (e.g. add a new super-class at run-time) is powerful but obviously increases the complexity of the underlying system.

The integration of the C and ProTalk languages is quite simple with the ability to mix the two languages inside the same method. While the ProTalk language is more concise and powerful, there are many operations for which one must resort to C. Examples of such cases are when the developer needs to access some facility in the underlying operating or file system. On the other hand, there are some operations which must be written in ProTalk - for example slot-formulae and trigger rules. Knowledge expressions are a very powerful feature of ProTalk. They can be used to construct forward or backward chaining rule sets forming the basis of expert systems. We have used knowledge expressions to perform searches on object instances in a manner that is similar to a database query language (e.g. SQL) but with the advantage that the constructs are part of the programming language.

We would prefer to be able to use a non-proprietary language (e.g. C++) instead of ProTalk but recognise that some features of ProTalk would need to be provided by other means (e.g. the ability to dynamically change the inheritance of an object instance, access to metamodel information, knowledge expressions etc.)

6.2.3 User interface construction

The GUI builder offers similar facilities to X-Designer but has one important advantage - the Data Linkage tool. This tool allows application objects to be linked to graphical objects without programming. The relation between the application and graphical objects is maintained so that, for example, if a new instance of an object class is created it can be made to appear automatically in a list widget. Similarly if a data linkage is established between an object’s attribute and an edit widget then modifications made by the user on the screen are used to automatically update the attribute’s value. To provide a similar behaviour with X-Designer the developer would be required to write many callback routines to move the values to and from the screen. However, we have found that we cannot integrate third party widgets so easily in the Interface Workbench as one can with X-Designer.
6.2.4 Persistent objects

The internal object persistence is very useful and has been used as the basis of the Emf database and online bookkeeping database. It is very simple to use with only one function call needed to save or restore a whole set of objects to or from disk. When used in this manner, the Object Diagrammer becomes a graphical database schema editor. The system also has the advantage of providing limited schema evolution - that is to say that objects can be saved to disk, the schema changed and the database re-read into memory. The facilities of the object persistence cannot be compared to a real database since it has no notion of transaction support, concurrency control, versioning or distributed access but it is more akin to an OO version of QUID.

6.2.5 Distributed applications

We have performed limited tests with the CommManager package for distributed applications but do not currently use it in any of our applications. It appears to work satisfactorily on a network of sun workstations but we have yet to test it in a heterogeneous (HP and sun) environment. The programming overhead of distributing an application over several processes is minimal. Conceptually, it can be seen as the object-oriented equivalent of remote procedure calls (RPCs) with the advantage than the developer is not required to define the interfaces via some specialized language (e.g as used by RPC compilers). As with RPCs, an application can be developed as a single process and distributed later with the minimum of disruption. We were particularly interested in combining the CommManager with the object persistence and object monitors in order to provide a distributed, reactive data store, whereby clients of a data server process could use monitors to be informed of any changes to the objects managed by the server.

6.2.6 Software distribution

We have shown that it is possible to integrate OMW with third party archive systems such as CVS. We have not yet had the opportunity to test the multi-developer extension to the tool which has been delivered as a beta-release with the latest version.

Because of the sophistication of the Kappa object environment, we can see the need for the run-time library. However, the run-time library restricts the platforms on which we can run our generated applications. This means that we cannot use OMW to develop applications which must run on our front-end processors (e.g. RAID boards running EP/LX or LynxOS). But since single VME board versions of the HP and sun workstations are becoming available, we can run OMW/kappa applications on these and still have access to the VME bus etc.

6.2.7 Robustness of the tools

There are a number of problems and bugs in the CASE tool that have slowed down the development of the applications. In general we have found that the interpreter is the source of many problems especially if the currently loaded application becomes corrupted. Often the only solution is to exit and restart the tool. If an application is saved while the tool is in this state then the application may itself be corrupted which means it cannot be reloaded in to the tool. One can try to repair such faults but it requires a detailed knowledge of the tool and Kappa environment, alternatively one must abandon the current copy of the application (possibly causing some work to be lost) and return to a backup copy (the tool does make backup copies whenever a new version of an application is saved). The tool requires a lot of resources in terms
of memory and swap space which restricts its use to the more powerful workstation configurations and means the startup-time for the tool is quite long.

7 Conclusions and future

Our experience has been very positive and we believe that there is a great future in software engineering and CASE technology for HEP, especially for large software production. The benefits in communications between the developer and the user, with provisions for project management and support for the full life-cycle including code reliability and maintainability are tremendous. The quality and quantity of commercial CASE tools is increasing rapidly and are sure to offer the most cost-effective and organized manner for HEP projects to develop their software. The software engineering field, and the CASE market in particular, are in rapid evolution and, although industry is investing effort in this direction, standardization is still far away.

We will continue to use OMW in the near future and other applications for the DAQ are already being constructed. In general one can say that OMW and OOIE provide nearly all the features we think we would need to develop software for LHC data acquisition systems but we would prefer a tool which:

- is easier to master (OMW's sophistication can be bewildering for a beginner)
- requires less resources to run
- provides support for finite-state-machines as a means of defining object behaviour
- allows the developer to write methods in a non-proprietary OO language
- generates code that does not require a run-time library from the software tool vendor

It is unlikely that all these requirements can be satisfied by a single CASE tool. OMW is by far the most comprehensive tool we have seen to date and we would like to use it as a basis for defining the features required from some future set of tools, probably from different vendors, which could form the software development environment for LHC DAQ systems.

8 Acknowledgments

The evaluation work presented in this paper involved several people including Carmen Maidantchik, Ashraf Patel and Igor Soloviev. A number of other people have also used the development environment and provided valuable input including Igor Gaponenko, Cecilia Magherini and Zuxuan Qian. Thanks go to Gottfried Kellner for contributing substantially to the purchase of the software tools mentioned in this paper and to ECP/PT group for providing a host site for their installation. It is a credit to the guidance of Livio Mapelli and Giuseppe Mornacchi that such technologies can be incorporated into the DAQ without disrupting it effectiveness.
9 References

Information on the RD13 project and the tools used during software development are available through the World Wide Web at URL http://rd13doc.cern.ch/. Information on OMW is available at http://www.intellicorp.com/.

CICERO RD38: A Distributed Information System for HEP Controls

J-M Le Goff
CERN, Geneva, Switzerland

R McClatchey
Dept. of Computing, Univ. West of England, Bristol, UK

Abstract

Large scale, distributed industrial and scientific systems share many technical problems in the implementation of their control systems. With the advent of standards in distributed object technology, it is now feasible to reuse control objects across control systems. The CERN CICERO project aims to use object-oriented methods to design the main building blocks of a generic control information system based on the distributed object standard, CORBA [1]. CICERO is producing an integrating environment (Cortex [2]) into which distributed user control objects will ultimately be 'plugged and played' and a supporting information system for the configuration and management of that environment. Cortex has been designed to be sufficiently generic in nature to allow its reuse by any future control system used at CERN and potentially by medium to large scale industrial control systems such as power systems, satellite control and telecommunications network management. This paper describes the main design concepts behind Cortex and the enabling technology and the software engineering methods used in implementing Cortex.

1 Introduction

The latest experiments in particle physics at the CERN laboratory in Geneva are composed of large numbers of sophisticated detectors and devices each potentially constructed by geographically separated development teams using disparate electronics and software. Each detector requires software to control the acquisition of data and the supervision and operation of great numbers of sensors and actuators. Control activities include monitoring devices, maintaining the safety of the experimental setup and the control of low level automation loops to automatically maintain devices in operational conditions. To reduce development costs physicists are looking for partially reusable solutions to their technical problems such as the incorporation of industrial products with home-grown products [3] for tasks in the control system. However, in the recent past it has proved difficult to integrate commercial products together (e.g. PLCs with VME) and to integrate these products with existing CERN-made control (sub-)systems. In essence, what is required is an overall framework to facilitate integration between control system elements. Such a framework (or software integration platform) should go beyond defining standard interfaces, it should guarantee that commercial products can exchange information and collaborate regardless of the organisation of the overall control system.

The next generation of CERN experiments at LHC - the Large Hadron Collider [4] - will involve collaborations of many tens of institutes and over 1,000 physicists, engineers and computer scientists from around the world. The knowledge required to construct and monitor the experimental (sub-)detectors will be distributed between these institutes making it difficult to impose standards. There will consequently be significant problems of information transfer to ensure that each (sub-) detector retains autonomy of control but can work with other (sub-)detectors for data-acquisition. In addition, the LHC detectors will be required to have a long lifecycle since the experiments will take data for several years and, as a consequence, maintainability will be an important consideration. The experimental groups will also be working to very tight time and cost schedules. As the experiments grow, so the control system should grow from an initial lab-based test system to test-beam operation and to the fully-fledged experimental system. The Cortex element of the CICERO project intends to provide an integrating scheme (Cortex [2]) to enable the building of distributed control systems where
Responsibilities are distributed amongst control elements that have to collaborate together.

Experience of the development of control systems for the LEP experiments [5] and anticipation of the increased demands which will be generated by the new and larger experiments for LHC has identified the main constraints for the design of such an integrating scheme. Firstly, since the LHC experiments will be equipped with > 100,000 sensors and activators, the management of control system complexity is a major consideration. Object-oriented design techniques embodying abstraction, inheritance and the use of classes and objects will aid the design here. Secondly the problem of concurrent and collaborative software engineering requires addressing. This is particularly true when the development of the control software is carried out by engineers who are separated geographically. Thirdly, the software developed should provide stability, flexibility and availability of control system elements so that system down-time (such as that for upgrades) is minimised. Finally, the control system software should provide balanced, distributed processing in the heterogeneous environment of High Energy Physics (HEP) control systems.

The next section identifies the characteristics of control systems as used in HEP and illustrates how the concepts of hierarchy and collaboration are important in control. The design philosophy behind the construction of Cortex is identified in the sections that follow showing how those design constraints identified above have been addressed. The enabling object-based technology and standards used in the design and development of Cortex are later investigated before the current status and future development plans for Cortex are described.

2 Control Systems Structure in High Energy Physics

From the point of view of control, a HEP accelerator or experiment can be visualised as a set of devices (muon detector, electromagnetic calorimeter, etc.) and systems (data acquisition system, gas system, etc.) performing specific functions and collaborating, by the exchange of control information, in order to achieve the accelerators’s or experiment’s objectives. Each of these devices and systems, are usually decomposed into subsystems, with specific functions, different information needs and different degrees of autonomy. These subsystems could be further subdivided, to the appropriate level of granularity required by the underlying hardware or by functional decomposition (Figure 1).

In order to properly specify the collaborative distributed control system, two basic features are required to be offered to the users of the control system. On the one hand, the users need to be able to organise hierarchically the different elements of the collaborative distributed control system. On the other hand, the user must be able to specify the collaboration needs between these elements, independently of their hierarchical organisation.

In the first case, the hierarchical organisation of the control system reflects the layout of the experimental setup, supports detector-specific global operations (e.g. for data taking or beam operation) and local operations (for calibration and test purposes e.g. ramp-up high voltage) and provides a representation which allows apparent complexity of the experiment to be hidden from the user. In the second case, the collaboration aspects provides the user with facilities to specify which information is to be shared between control elements and how that information is to be shared. This includes definition of the formats of shared data or services and the form of transport of the data. Two major mechanisms for data transfer and service invocation are required: multicast communication which can be used for functions such as start calibration or the distribution of monitoring data from a single producer to multiple consumers and point-to-point communication which can be used for interlocking front-ends or the dispatch of a command for invocation between a requester and a performer (e.g. operator asks the muon system to ramp down its high voltage to 2 KV).

All the above needs can be abstracted by two orthogonal relationships which must be satisfied by any system used for controls in HEP:
• the “composition” relationship which describes the hierarchical organisation of control elements to support global behaviour. For instance, the “Experiment” element is composed of the “Muon detector”, “Run control”, “Gas system”, “Cooling and ventilation”, “Security”, “Tracker”, “ECAL”, and “HCAL” elements and the “Muon Detector” itself is composed of “High Voltage”, “Low Voltage”, “Discriminator System”, “TDC” and “Alignment System” elements and

• the “collaboration” relationship which is the hierarchy independent communication between elements to support dedicated information and command exchanges. For instance, the “Muon detector” element is collaborating with the “Gas system” element.

![An experiment hierarchy](image)

**Figure 1:** An experiment hierarchy

### 3 Scope and Constraints in Cortex

HEP experiments and accelerators are normally developed by various teams with specific task assignments. Usually, the different parts of a HEP experiment are developed by different institutes in their laboratories and gathered together later after being tested. The experiments are tuned up at CERN before new tests, or calibration and operation. For instance, the gas system and the high voltage system of a muon detector may be produced and tested by two different teams. Cortex must support an equivalent life cycle for the collaborative distributed control systems of these experiments or accelerators. In particular, it should be possible to develop control systems and later to integrate them in a collaborative distributed control system without major code modifications. Users should only have to configure the existing collaborative control system, by introducing their part of the development and specifying the information and commands to be exchanged with this part in the global control system.
Although the different elements of a HEP experiment are operated globally during data taking, sometimes some subsystems or devices have to be operational while some others are not. For example, during calibration and test, the muon detector and the gas system can be operational while the magnet is switched off. Cortex shall promote the modularity of distributed control systems in order to allow some integrated control systems to be operational while some others are not. Malfunctioning hardware may lead control elements to send incorrect information to other elements, which can consequently take inadequate actions. If the hardware cannot be repaired immediately, Cortex shall allow operators to prevent this incorrect information from being broadcasted. Cortex will also support users during the reengineering phases of the integrated control systems by allowing them to reuse and integrate existing software.

Cortex is required to support the entire collaborative distributed control system life cycle by:

• providing a multi user development environment,

• supporting the testing and simulation of users’ control elements with respect to the integration within the Cortex control system,

• offering tools to operate and protect the control system from faulty processes implementing some control elements,

• allowing the integration of already existing control systems,

• allowing collaboration with other autonomous control systems.

Cortex intends to provide a control system designer with the ability to integrate his particular control system efficiently and with minimal cost and effort.

The architecture of any distributed control system will change during the lifetime of the accelerator or the experiment it is operating. The control system integrating platform, Cortex, must therefore support a mechanism to allow a new version of the distributed control system to be in preparation while an older one is operated on-line. It shall also support the backup and the restore of a given version of the collaborative distributed control system. Furthermore, tests and validation (and possibly simulations) of a new configuration will be needed before it is applied to the operating on-line system.

4 The Cortex Design Philosophy

• The above constraints have led to a so-called dual-face approach (see figure 2.) being taken in Cortex:

• an off-line Cortex representation is required to handle the logical descriptions of the architecture of the distributed control system and to describe the various information and commands to be exchanged between the different control elements. This so-called Repository also holds the description of the hardware model from which the on-line distributed control system is constructed, and

• an on-line Cortex representation is also required through which the control elements can exchange information and commands in a pseudo- ‘plug-and-play’ fashion. Control elements operating within the Cortex Infrastructure can access the Cortex Repository through this so-called Infrastructure. A generation mechanism is provided to facilitate updates of the on-line Infrastructure according to the Repository contents.

The responsibility of Cortex is twofold: on the one hand it has to support the description of the architecture of the distributed control system and the definition of the information and commands to be exchanged between the control elements. The off-line representation of the control system therefore addresses the issues of the management of control system complexity and that of concurrent and collaborative software engineering identified earlier. On the other hand, Cortex must transport and distribute these data and commands to the appropriate
control elements when part or all of the distributed control system is in operation. This distribution must be independent of the number of hardware elements on which the various control elements are operating. The integrating framework must be flexible enough to support the addition or removal of control elements, without deteriorating the operation of the rest of the distributed control system. The on-line Cortex Infrastructure thereby addresses the demands of stability, flexibility and availability of control system elements and that of providing balanced and distributed processing for the control system.

Figure 2: The dual-face approach

5 Cortex off-line

5.1 Components

Off-line a Cortex control system is visualised as a collection of collaborating components. These components map onto those on-line processes in a control system which produce or consume information and they can be of various kinds. They may or may not be “control components”. For example gas systems or high voltage systems are managed by software that are control components. Each has a read-out system and has responsibility for the hardware. Indeed some control components may need real time capabilities, for instance, elements interfacing front-ends performing interlocks. On the other hand, an on-line documentation element or a user interface element are “non-control elements” - they do not have responsibility for the hardware. They may, however, support high level control functionalities such as alarm filtering, user assistance, preventative maintenance etc. Cortex can therefore be used to integrate software which implements facilities common to any kind of control element such as loggers, archivers, retrievers, DUls, final state machines, etc.

5.2 Compositeness and Collaborative Groups

Compositeness is the mechanism proposed in Cortex to support the logical encapsulation of a distributed control system. Components may be composed of smaller components. Such components are referred to as composite components and are often used to separate functions or to provide the granularity required by the underlying hardware (figure 3). Users must be able to operate the complete control system from a global standpoint or operate each component independently via the composite components. Additionally, users must be able to specify communication requirements at any level of component, regardless of the inherent hierarchical organisation. In practice then a composite component may be chosen either:
• to represent a global control process when it will support global operations such as those to start/stop the global control process or
• to represent a local control process when it will support operations such as start/stop for any specific encapsulated component.

Grouping is a complementary mechanism to Compositeness proposed in Cortex to allow specification of information and command exchange at any level of granularity. A collaboration group is composed of a set of components that make available certain information and services to the other components in the group. Two components will be able to exchange information and commands if and only if they belong to the same collaboration group. Components can be part of more than one group. Collaboration groups can be established across encapsulations of sub-systems. The combination of compositeness and collaboration groups allows the user to refine and optimise the communication at an appropriate level of control system component.

5.3 Publishing and Subscribing

Within a collaboration group, a component can provide information to other components by publishing items (data or services). If granted permission by the publisher, any component of a collaboration group (other than the publisher) can access this information by subscribing to the published items. The set of published and subscribed items handled by a component within a collaboration group is called a component interface. A component usually has a different interface for each collaboration group in which it is a member.

A component can subscribe in one collaboration group and republish in another collaboration group. The publish and subscribe mechanisms are fully asynchronous. In particular, an item can be depublished while it is still subscribed by other components. If this situation is propagated to the on-line system, the corresponding on-line Infrastructure will support such incon-

![Diagram](Figure 3: (Composite) Components & Grouping
Key: Groups are represented by ellipses, Components by rectangles and Component interfaces by shaded boxes. Composite Components are shown as double rectangles.)
sistencies and inform the appropriate subscriber components. Compositeness and collaboration grouping are the mechanisms for supporting information abstraction in Cortex: some of the collaboration groups can be organised to enforce encapsulation of sub-systems (so-called strict encapsulation). This will allow composite components to subscribe within their encapsulation and publish refined information to other composite components in other collaboration groups.

6 Cortex on-line

The Cortex on-line Infrastructure is a set of entities responsible for the distribution of information and for the transmission of commands (service invocation) to the appropriate components, according to the Cortex Repository specifications.

The Cortex Infrastructure provides two selectable ways of data exchange between components. The push mechanism which allows components to push new information into the Infrastructure or to receive information from the Infrastructure and the pull consumer mechanism which allows components to retrieve information from the Infrastructure at their convenience. The push mechanism is recommended for security information such as alarms. The pull mechanism is more suitable for monitoring components offering refreshed information upon users’ request. In both cases, only information specified in the Cortex Repository will be transported and delivered to the appropriate components. Version inconsistency between the information sender and the information receiver are handled by the Infrastructure at message level. For example, a component may pull from the Infrastructure items which are no longer published (in the Repository). The dynamic information contained in these items is no longer refreshed and the corresponding component will be informed. Time stamps will contain the last time these items have ben refreshed.

The on-line architecture supports a separated set of messages to handle service invocation called command messages. Commands are persistent in the Infrastructure from the moment they have been issued until they are completed (successfully or not) or refused by performers. Two basic types of services are available. Firstly services can be cancellable or non-cancellable: a requesting component can cancel its request while the component offering the service is processing the command. Secondly services can be multi- or single-requestable: more than one requesting components can issue a command to the same performer component.

More than one requesting component can invoke the same single-requestable command hence addressing the same performing component. The on-line architecture handles possible access conflicts using an internal protocol based on locking. Commands are not direct implementations of operations in the OMG Object Model, which do not support some specific control functionalities such as authentication, availability and progress report features.

7 Enabling Technology

In an effort to maximise the reusability of code in Cortex and to provide an incremental development route, object standards have been followed in developing Cortex. In particular the Cortex on-line Infrastructure is based on the Object Management Architecture (OMA) and the Cortex off-line Repository on the Object Database Management Group (ODMG) standard. The following sections introduce these standards and identify the important aspects of the standard-gauge Cortex. The use of the OMA standards are considered in the next section of this paper where reusability of control system components is considered.

7.1 The OMG Object Management Architecture Standard

The Object Management Group (OMG) is an industry consortium dedicated to creating object management standards necessary to achieve the goal of interoperability between heterogeneous-


ous, distributed object based systems. The Object Management Architecture (OMA) [6] is the reference model for the OMG standards and identifies three major categories of specifications or architectures:

- **Common Object Request Broker Architecture (CORBA)**[7]
- **Common Object Services Specification (COSS)**[8]
- **Common Facilities Architecture (CFA)**[9]

In the OMA object model, objects provide services to clients as shown in figure 4. An interface is a description of a set of the possible services that a client may request and is specified in Interface Definition Language (IDL). Clients are not written in IDL but in the implementation language for which mappings have been provided. IDL mappings are currently available for C and C++.

The initial focus of the OMG was on the specification of the Object Request Broker (ORB) as detailed in the Common Object Request Broker Architecture (CORBA) document. The ORB forms the key part of the architecture by describing how objects are located, operations accessed, and arguments passed transparently and flexibly such that integration of a variety of distributed object based systems can be achieved. CORBA specifies IDL, the object invocation interfaces together with an interface Repository on the client side, object adaptors (OA) on the server side and the ORB core. It is the ORB core which does the object location, message delivery and binding between clients and object implementations. There are two ways that a client can make an object invocation request, one is through a static interface and the other through a dynamic interface. Whichever invocation interface is used a target object cannot tell which method was used. The important difference between the invocation interfaces will be the amount of time taken to invoke the implementation object.

The actual object implementations are constructed from programs which could be executable scripts or loadable modules. An implementation object can be designed so that a single program implements the whole objects interface behaviour or one program can be used to implement each of the methods. The method binding interface to the ORB core is through skeletons. However, the object implementation interacts with the ORB core in a variety of ways, for example, to register itself, to request an object reference and to invoke ORB serv-

---

**Figure 4:** The OMG/CORBA Model for CICERO.

Application objects: Cortex Infrastructure objects.
Common facilities: OODBMS, Real-Time controls.
ices. This interface is provided by the Object Adapter (OA). The OA defines most of the services from the ORB core that the Object Implementation can depend on such as activation, deactivation and access control to object implementations. Different ORBs and different operating environments will provide different services and levels of service, if the ORB service is available then the OA simply provides an interface to it and if the service is not available then the OA must provide it. The CORBA specification suggests that it is not necessary for all object adaptors to provide the same interface or functionality an example of a special OA interface would be one that connects to objects stored in an object-oriented database. There is one OA that all CORBA implementations must provide and that is the Basic Object Adapter (BOA). The BOA is concerned with activation, deactivation and access control of object implementations and thus the main responsibilities are:

- Generation and interpretation of object references
- Authentication of the principal making the call
- Activation and deactivation of the implementation
- Activation and deactivation of individual objects

CORBA provides the basic communication channel through which objects interact however the system services it provides are limited. Fundamental system services such as naming, replication, security, transactions and time are key to building distributed applications. The Common Object Services Specification (COSS) defines in terms of interfaces and objects a collection of these services. COSS volume 1 covers Naming which provides the ability to attach textual names to object references, Event Notification which provides an event notification service for unexpected events, Life Cycle Support for creating, deleting, copying and moving of objects and a Persistent Object Service. COSS volume two which is expected in mid 1995 will specify transactions, externalisation, relationships and concurrency services. COSS Volume 3 will feature Security and Secure Time Service and COSS Volume 4 will include Object Query and Object Properties services. These later two volumes are expected later in 1995.

The Common Facilities Architecture (CFA) is the third and final area of OMA to be defined (interfaces and objects) and is a collection of higher level services which may be broadly applicable to many applications. Four major horizontal domains for such facilities have been identified so far: User Interface, Systems Management, Information Management which covers the modelling, definition, storage, retrieval, management and interchange of information, and Task Management which covers the automation of work. Vertical Market Facilities represents technology that supports various vertical markets such as retailing, telecoms, CAD or health care. Each Common Facility interface is defined in IDL and as such may inherit behaviour of the more fundamental Common Object Services. Similarly implementers building distributed applications can make extensive reuse of Object Services and Common Facilities by OMG IDL based inheritance.

### 7.2 The Object Database Standard (ODMG)

The Object Database Management Group (ODMG) has put forward a set of standards allowing an Object Database Management System (ODBMS) user to write portable applications, i.e. applications that could run in more than one ODBMS product (the schema will be portable as well as the application accessing it). The proposed standard will, eventually, be helpful in allowing interoperability between different ODBMS products, allowing the development of distributed heterogeneous database communicating through the OMG Object Request Broker.

The ODMG defines an ODBMS to be a Data Base Management System (DBMS) that integrates database capabilities with object-oriented programming language capabilities. An ODBMS makes database objects appear as programming language objects, extending the language with transparently persistent data, concurrency control, data recovery, associative que-
ries and other database capabilities. The major components of an ODBMS are the following:

- the Object Model. This is based on the OMG Object Model. An ODBMS Profile has been created as an extension of the OMG Core Object Model to support the ODBMS specific needs (e.g. Relationships),
- the Object Definition Language (ODL). This is the data definition language for the ODBMSs. It is based on the OMG Interface Definition Language,
- the Object Query Language (OQL). This is a declarative language for querying and updating database objects. It is based on the relational standard SQL,
- the C++ Language Binding. This explains how to write portable C++ code that manipulates persistent objects. It is called the C++ Object Manipulation Language. The binding also includes a version of the ODL that uses the C++ syntax, a mechanism to invoke OQL and procedures for operation on database and transactions.
- the SmallTalk Language Binding (in preparation).

The current version of the standard is ODMG-93. ODMG-95 is in a draft version. ODMG-93 foresees the ability to access an ODMG-compliant product through an ORB using a special Object Adapter for this ODMG product. Users have to design this adapter or intermediate objects that can make use for instance, of C++ OML and of the OMG C++ Language Mapping.

7.3 Mapping Enabling Technology onto the Cortex Design Philosophy

In Cortex, an ODBMS is being used as the vehicle for the off-line Repository to support a standardised access for the CORBA objects in the on-line Infrastructure. The Repository has been designed to support the notions of Compositeness and Collaboration Groups, Publishing and Subscription and Components as described in an earlier section. ODBMSs provide persistence of object information, and all the advantages of DBMS systems such as version management, concurrency control, security and recovery. This enables control system designers to save a full description of the experimental setup in an object base and to modify that description over time as the experiment grows.

The Cortex on-line Infrastructure is instantiated as a set of CORBA objects responsible for the distribution of information and for the transmission of commands to the appropriate components, according to the description resident in the Cortex Repository. On-line objects in Cortex are written in C++ and use Iona Technologies implementation of CORBA, called Orbix, for object location, access and communication services. The physical location of the components and the Infrastructure will depend on the hardware setup available. This setup may evolve with time for performance reasons or for maintenance purposes. In these cases, the system functionalities must be maintained when part of the hardware is changing. This operation should take place without disturbing the operation of the parts of the distributed control system which are not involved in this upgrade. The location transparency is fully supported by CORBA. CORBA makes no provisions for message sender identification. Any program can potentially send a message to a CORBA object. To avoid unpredictable overloads, Cortex provides an authentication mechanism to ensure that for any new starting process is effectively representing a component known to the Cortex Repository.

As an example of the use of the OMA standard in Cortex the next section investigates how Cortex provides reusability both of control system components and complete control sub-systems.

7.4 Reusability and Cortex

The basis for re-use in the CICERO project is through the use of CORBA IDLs and ORBs. Reusability can be exploited in CICERO both at a level internal to components and at a level
external to components. Firstly at the internal level, consider the re-use of component code as the hardware of the control system is evolving. By using IDL stubs for client invocation and an IDL skeleton to package the existing (component) code for CORBA compliance, the control system can be allowed to grow whilst reusing existing components. As an example, consider the incorporation into a UNIX-based control system of an existing data logger which runs on VMS/ORACLE. To reuse this component it is necessary only to build an IDL interface to this logger, using an ORB which supports VMS. Then a UNIX component will be able to send messages such as store and retrieve through this interface to log information without having to know the complexities of UNIX-VMS translation. In addition, such a component will be able to log any additional Cortex messages issued by any other existing components according to the description stored in the Cortex Repository.

Using CORBA IDL for interface specification permits the sub-division of a large software module into smaller, easier to manage units with simpler functionality. This facilitates reusability in that it supports:

- the evolution and partial upgrade of complex control systems and
- the re-use of existing (legacy) systems through CORBA objects.

In this example of reusability, partial re-engineering of component code is again required since the IDL specification takes place inside the component code.

CICERO is also able to exploit reusability at a level external to components. This can be achieved through the use of so-called Reusable Components. These can be regarded as templates for components with logical input/output which can be instantiated as many times as there are physical devices. At the time of instantiation, there may be no hardware assignment - only at the time of assignment will the desired functionality become apparent and the component code reused. For example, consider a 16-channel Analogue to digital Converter (ADC) read-out component coupled with an ADC Converter component. The first component is hardware independent, except for the ADC gain, and can be reused as many times as ADCs are needed in the system. The second component is context dependent and can evolve with the hardware of the system. If the second component is data driven and obtains its configuration data from the Repository, then the ADC/ADC Converter pair can be reused with no code modification in any subsystem of the experimental setup. Cortex offers the possibility of decomposing a complex control system into data acquisition components, command components and automation loop components offering the possibility of reusing or sharing components in different control systems.

Another example of reusable components is possible in CICERO when whole sets of components are reused at the control system level. Here, consider two Cortex control systems developed independently and merged at a point in time. For example, a development or test setup which has been locally setup and then physically moved to be made part of (integrated with) a larger control environment. In this case since there is compliance at the Cortex level, no code modification whatsoever is required and the integration takes place through the addition of an intermediate component which resolves the match between the two Cortex systems. That is, the intermediate component subscribes to the items of the test control system and republishes the converted items for the larger control system. In addition, it is possible to split the two control systems for independent running at a later point in time without code modification.

8 Software Engineering Standards in the Development of Cortex.

The previous sections noted the object-based standards adhered to in the development of Cortex and investigated reuse of control system components based on these standards. This section describes the methods that were followed in implementing Cortex and in particular concentrates on the software engineering techniques used which enabled the development of Cortex.
8.1 The ESA PSS-05 Standard

As HEP systems become more complex, the need for rigor in software engineering increases in importance. In addition, as the development of these complex systems is increasingly carried out remotely from CERN or by developers on short-term contracts at CERN, the need for clearly defined deliverables and interfaces between (sub-) systems also becomes crucial. As a consequence, software engineering standards have been investigated in the last few years by large groups of developers in HEP. Work at the European Southern Observatory [10] and the European Space Operation Centre [11] into standards has recommended the use of the European Space Agency’s software engineering standards alongside those from the IEEE and IEE.

The European Space Agency Procedures, Specifications and Standards [12] method is an essential feature of CICERO. This standard has been developed for the European Space Agency to ensure that any project has the best chances of a successful outcome. There are two major parts: the products themselves and the procedures to produce them. The products are the documents and software used to create, use and maintain software. The procedures guide the system developer in project management, software configuration management, software verification and validation and software quality assurance. The documentation for this standard includes both mandatory and optional sections, divided into three levels. In order to meet the ESA standard, all mandatory operations must be carried out or documentation produced as appropriate. The process of production is divided into six phases, following the standard waterfall life-cycle model of User Requirements Definition, Software Requirements Definition, Architectural Design, Detailed Design and Production, Transfer and Operations & Maintenance. In addition there are documents on Structured Analysis, Fortran Coding, Ada Coding & C Coding standards.

Management of the software lifecycle is catered for in the ESA standards through the use of a Software Configuration Management Plan (SCMP), a Software Verification and Validation Plan (SVVP) and through Software Quality Assurance (SQA). These plans are detailed in [12] and provide the project manager with requirements for identifying, controlling, releasing and changing software releases and for recording their status. The SVVP provides for the review, testing and auditing of the delivered software products. Further, the project manager can ensure quality is being maintained in software delivery by following the recommendation of the SQA plan.

The ESA standards were originally based on the ‘Waterfall Model’ of the software lifecycle, following a phased approach to software development. Modified forms of this approach include the incremental delivery and evolutionary development models. In the development of Cortex an evolutionary approach has been adopted which overcomes some of the limitations of the waterfall model. The evolutionary approach allows for the planned delivery of multiple releases of Cortex, with each release incorporating the experiences of earlier releases in a manner analogous to the ‘Spiral Model’ of software development suggested by Boehm [13].

8.2 The OMT Methodology

The CICERO project has selected the European Space Agency standard PSS-05-01 [12] as the life-cycle model to support the development of its software. Within that framework more specific software engineering methods are being used. Foremost amongst these is the object-oriented design method developed by James Rumbaugh and colleagues, OMT [14].

The OMT method comprises a number of models which are developed and enhanced as the project moves from requirements analysis through design to implementation. There are several CASE tools available for generating OMT diagrams, but, since it is not a standard but a methodology, the tools need to be embedded in a full standard such as ESA. OMT involves several stages, and is in some ways an extension to the Entity Relationship approach for designing and documenting systems. It is based strongly on the relationship between objects.
and nouns in a textual description and on the behaviour of the object and the verbs to analyse the requirements.

Of widest use on the CICERO project have been the Object/Class model (a form of Extended Entity Relationship model showing static data relationships in the software), event traces (showing sequences of messages sent between objects to accomplish a given function) and the Dynamic Model (State Charts to define the temporal ordering of events impinging on a given object). These models have been supported by the use of a diagramming tool, Select. In addition, the CICERO project has used a code generation tool, OBLOG, which supports the OMT methodology, to generate User Components (in C++) from the OMT Object/Class model. The OBLOG tool integrates concepts from semantic data modelling and concurrent processing, employing objects as a unifying concept and aiming at a conceptually seamless methodology from requirements to implementation [15].

9 Experience of Applying Object Technology in Cortex

Having identified the technology and software engineering techniques used in the development of Cortex, conclusions made be drawn in their use.

9.1 Use of OMA standards

The main conclusions that can be drawn in Cortex from the experience of the use of OMA and CORBA are that:

• CORBA has successfully demonstrated the ability to allow the designers to wrap up existing legacy software systems, some of them shell scripts, as CORBA objects and integrate them into the Cortex system.

• the CORBA interface inheritance has allowed reuse of object behaviour.

• considering that the collaboration team is spread across the far corners of Europe and beyond with each group working on its own part of the prototype the “plugging” together of the components using CORBA to construct the prototype was a notable success.

• the CORBA tools (Orbix) have caused a few minor problems particularly time-out of object connections. However most of these problems have appeared to be resolved in the more recent version 1.3.

• none of the OMG fundamental object services (COSS) have been implemented as yet, in fact most of them have yet to be fully specified. This presents a dilemma as the CICERO team has proceeded to the design stage and have commenced specifying and designing its own object services. It will be interesting to see how long it takes for the OMG object services to be bundled into existing commercial CORBA Orbs.

• similarly the OMG Common Facilities Architecture has only just been published. However this document is intended more as a management tool intended for controlling development of and positioning Common Facilities and their specifications. The experience of using the OMG OMA within the CICERO Project may well provide the basis for the collaboration team to submit suggestions to the Common Facilities Task Force as to candidate common facilities within the control systems vertical market.

9.2 Use of ESA and OMT

The use of the ESA methods during the prototype development undoubtedly helped to clarify the design of the first release of Cortex. Such use of the ESA standards requires experience, especially when identifying the boundaries between the different phases of the lifecycle. In principle, ESA standards do not enforce any particular methodology. In practice, however, the ESA guidelines are clearly supporting a functional breakdown of the problem statement and
hence implicitly imply non object-oriented methodologies. Some trade-offs had to be made to support the OMT methodology, especially in the Architectural and Detailed Design phases of the ESA standards. One example is in resolving the relationship between the demands on the documentation produced in accordance with the ESA standards, particularly in the prominence given to natural language in the User and Software Requirements documents and the need for more precise semantics demanded by the OMT modelling approach. Guidelines are needed to help in the integration of these two standards.

Software produced for a pilot project is similar to production software with respect to robustness and reliability. Therefore management guidelines supporting the software lifecycle as enhanced by the ESA-PSS 05 standard must be strictly followed. In particular, as stated earlier, a complete set of Software Project Management, Software Configuration Management, Software Verification and Validation and Software Quality Assurance plans must be produced in parallel with the software lifecycle documents.

The OMT notation is strong in describing the abstract design but is lacking when describing the implemented system. This failing was felt during the Architectural and Detailed Design phases yet it is here, when the volume of design information increases substantially, that notations to capture the design and tools for their manipulation are essential. Work on design notations and the capture and re-use of designs is the focus of some researchers in the OO field and the results of this work should be made accessible to the CICERO team. OMT is not particularly targeted at the design of real-time systems and is not sufficient to handle concurrency. As a result, the treatment of some issues of concern in the development of Cortex are inadequately addressed. Some of these issues are better handled by later developments of the OMT approach, notably the Syntropy method [16]. Syntropy provides additional rigor to the OMT method, particularly in the design of cooperating processes.

It was clear during the prototype development phase that in normal discourse about Cortex, designers and users make great use of concrete examples of usage of the system and specific, telling, examples of message sequencing within the implemented system. It is felt that the efficiency of the design process and of communication of the design to the users could be improved by the incorporation of scenarios and use cases into the formal documentation and design process following the Jacobson approach [17].

The quality of tools support for methods remains an issue for the Cortex developers. Platform specific tools, difficult access to design repositories and the lack of support for multi-user and multi-site development, lead to a loss of efficiency. In particular, it generates additional conversion and cross-checking tasks. Managing versions of documents, supporting relationships between models from different viewpoints on the design and tracking the process of design decision making are all desirable in the selected tallest. As yet there is no obvious single tool for supporting the use of OMT in developing control systems. Varsamidis et al. [18] advocate the combination of individual software tools, each specialised in a particular aspect of the design process for control systems. Tool evaluation, selection and guidelines in their use in an integrated manner remain tasks for the next phase of the Cortex development. The use of CASE tools like OBLOG for the automated generation of CORBA compliant component code will be investigated further, especially to increase the generated codes performance, to deal with more complex data types and to support concurrent access.

10 Closing Comments

The CICERO project was approved as a CERN research and development project (RD-38) in February 1994. Since then the project has grown and continues to attract further commercial and academic research interest. Following the ESA standards, the Cortex element of CICERO has gone through preliminary User and Software Requirements specification followed by Architectural and Detailed Design [19] for a demonstrable prototype.

The Cortex approach of integrating processes in its Infrastructure, allows for the abstraction
of control system information through the encapsulation of the underlying system components. This provides for ease of interfacing between the active objects in a Cortex control system and facilitates the provision of standard software modules to perform the activities of communication and control. In addition, this abstraction enables the generation of such code automatically from the description of the system in the off-line Repository.

CICERO is not only addressing the re-use of existing control system software, it is also more widely addressing the reuse of existing control and automation facilities or functionalities. It offers a smooth Infrastructure to support scalability and partial software and system re-engineering.

Within a year, the CICERO collaboration has demonstrated that it was possible to build an heterogeneous distributed control application using Object Oriented techniques (OO programming languages, CORBA and OODBMS), commercial products and high level functionalities like alarm filtering, user assistance and on-line documentation to help the user operate the control system [20]. Much work remains in CICERO Phase II to provide a system which could be used in practice. This phase is expected to span a period of 22 months starting in March 1995. It is expected that CICERO Phase II will reach this goal and will prepare the ground for a final consolidation phase yielding a set of software building blocks to enable the implementation both of LHC experiment control systems and industrial complex control systems.

11 Acknowledgements

In a collaborative project such as CICERO RD38, each collaborator deserves the thanks of the authors. Rather than cite each contributor thanks are extended to members of RD38 from BARC (Bombay, India), CERN (Geneva, Switzerland), CIEMAT (Madrid, Spain), IVO International (Helsinki, Finland), KFKI (Budapest, Hungary), OBLOG (Lisbon, Portugal), SEFT (Helsinki, Finland), SpaceBel (Brussels, Belgium), UID (Linkoping, Sweden), USDATA (Dallas, USA), UWE (Bristol, UK), Valmet Automation (Tampere, Finland) and VTT (Oulu, Finland). Special thanks go to those involved with the development of Cortex.

12 References


Object Oriented Programming and High Energy Physics

David R. Quarrie

Lawrence Berkeley National Laboratory, Berkeley, CA, USA.

Abstract

This paper discusses some aspects of the use of object oriented programming in high energy physics. The first section covers the goals and some fundamental concepts and asks whether OOP is merely a refinement of previous experience or whether it is a radical paradigm shift. The following sections illustrate the use of OOP through the use of some case studies. These focus on identifying the underlying abstractions and deriving a class hierarchy from them. Finally the support for OOP concepts that the C++ and Eiffel programming languages provide, as used in the case studies, is discussed.

1 OOP in HEP: Evolution or Revolution?

1.1 What is OOP?

Object oriented programming (OOP) is an approach to software development that encompasses all phases, from analysis through design and implementation onto testing and maintenance. I will use the definition from Booch [1]:

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

Unfortunately this definition uses terminology and concepts that are specific to OOP itself and which will be discussed later. However, it does not restrict itself to issues of programming language, but addresses the complete development cycle.

1.2 The goals of OOP

In the conventional model the software development cycle starts with requirements analysis, continues with a detailed design that addresses these requirements, then coding of an implementation that meets the design goals and finally testing and maintenance take place. This waterfall model is based on the underlying assumption that our knowledge during the requirements analysis phase is complete and the resulting implementation is coupled very tightly to that understanding. Variants of this model support the notion of some iteration between adjacent phases, but the basic principles remain unaltered. The resulting software is very tightly tied to our original understanding of the problem which is assumed to be complete.

Unfortunately experience teaches us otherwise; at the beginning of a project we almost certainly don’t really understand the full implications or scope. In a survey of several hundred software projects [2], 42% of the maintenance costs (changes in the product following initial release) resulted from unforeseeable changes in the user requirements. A further 18% of changes resulted from changes to the data formats.

We frequently experience such changes within the HEP environment. Most modern detectors undergo some upgrade to their capability during their lifetimes, or their physics goals change, perhaps as a result of improvements in the accelerator luminosity or other outside influences. The goal of our software should therefore not only be correctness, the ability to
match the requirements and specification, but should also be flexibility or extendibility, the
ability to be adapted to changes in the specifications.

Another goal is to produce software that is reusable because it is not tightly coupled to a
particular project. Reusability can lead to significant improvements in overall productivity, not
necessarily for a single project, but by applying the same software across multiple projects.

1.3 Fundamental Concepts

The following concepts are fundamental to OOP and the object oriented approach.

- **Objects** are entities that combine the attributes of both data and code and which can be
  manipulated or changed. They thus embody state and have state variables or attributes.
  They also exhibit an interface that describes how they may be manipulated. Finally, when
  they are manipulated they should do something useful; they exhibit behaviour. Note that
  it is what they do that is important, not how they do it. Their inner workings should be of
  no concern to us, as long as their overall behaviour conforms to the specifications. The
  interface to a clockwork stopwatch is essentially identical to that for an electronic one and
  as long as either performs correctly, it is irrelevant for us to know the details of the internal
  springs and levers or electronics components. In summary, an object has attributes to de-
  scribe its state and operations to describe its behaviour.

- **Abstraction** is “the elimination of the irrelevant and the amplification of the essential” [3]. The stopwatch is an abstraction that eliminates all details about its internal workings. We can start it, stop it and reset it. All stopwatches obey this abstract interface. Anything else is irrelevant. The focus should be on what an object does, not how it does it.

- **Encapsulation** or **information hiding** is a related concept to abstraction whereby portions
  of an object are hidden from the user. This results in an overall simplification since the
  programmer only has to deal with the interface or visible portion of the object, but also in
  additional security since the object is protected from inadvertent tampering. Thus the pos-
  sibility of side effects from modifications are greatly reduced and the software becomes
  both more amenable to change, and more robust under conditions of change. The pro-
  grammer wishing to make a change needs to understand a smaller portion of the overall
  system than would otherwise be the case. Finally, problems are easier to localize because
  the possible causes of corruption are reduced.

- **Classes** form the specification of an object. An object is an instance of a class or an in-
  stantiation from this specification. Objects that share the same structure and behaviour be-
  long to the same class.

- **Inheritance** is the key concept that distinguishes object oriented from object based sys-
  tems. In conjunction with abstraction it is the main provider of the extendibility and flex-
  ibility of OO systems. A class can be a specialization or extension of another class. A
  “dog” is a specialization of a “mammal”, which is itself a specialization of an “animal”. This
  is the inheritance relationship and is perhaps where the most confusing terminology
  appears. The parent class is sometimes called the ancestor or superclass, whilst the child
  class is sometimes called the descendant or subclass. Single inheritance is the situation
  when a class has a single direct ancestor, whilst multiple inheritance describes the case
  where a class has two or more direct ancestors (which might themselves have other an-
  cestors).

- **Polymorphism** describes the relationship between classes that have identical interfaces
  but different behaviours. A favourite example of this that is often quoted is a collection of
drawable objects, each of which has a draw operation, which performs the appropriate
drawing operation, a circle, rectangle etc.

- **Genericity** is the term used to describe parameterized interfaces. Thus a push-down stack
may contain integers or floating point numbers, or other quantities. However, all stacks
exhibit the same interface, they just operate on different data types.

- **Object Lifecycle Management** is the term used to describe the ability to dynamically create
new objects or destroy them. A related concept is that of garbage collection; the ability to
automatically destroy objects that are no longer referenced by any other objects and are
hence no longer needed.

- **Relationships** can exist between classes. A client-server or “uses” relationship is one
whereby one class, the server, provides services to another class, the client. A containment
or “has-a” relationship is one whereby an instance of one class is held by an instance of
another class. Containment is used to build classes that are composites of other classes.

### 1.4 The OOP Development Cycle

At the coarse level the object oriented development cycle is similar to that of the conventional,
structured approach. They both begin with a requirements analysis phase, followed by design,
implementation and finally testing and maintenance. However these phases both differ in their
details and the iterative development cycle is different. OOA consists of identifying objects and
their relationships from the problem domain. Structured Analysis consists of identifying the
data items and the processes that act on them. OOD creates an architecture for the
implementation by extending the identification of objects to the solution space, identifying
common policies for error detection and handling, memory management and generalized
approaches to control etc. Structured Design identifies a hierarchical decomposition into
modules that perform specific operations.

One major difference between the two methodologies is the emphasis that OOP places on
incremental development, sometimes called rapid prototyping. This has several advantages
including the early and extensive testing of major system interfaces, obtaining early feedback
from end users and the ability to see early results from a working system. Another strategy is
the identification of class categories or clusters that have a close internal relationship.
Development of such clusters can proceed in parallel, following their own internal development
and testing cycle. Thus they themselves are subjected to their own iterative development,
resulting in more robust software at the completion of the project. The concept is illustrated in
Figure 1.

![Diagram of Iterative Class Cluster Development Cycles](image-url)

**Figure 1:** Iterative Class Cluster Development Cycles
Another difference between the OOP development cycle and the conventional cycle is that, once every class cluster is essentially complete, an attempt should be made to see whether any new abstractions can be identified from commonalities across the clusters. This identification can modify class hierarchies and produce further simplification for possible reuse.

1.5 History of OOP

Object oriented programming came into being in 1967 with the design of the Simula programming language [4] which was an extension of the Algol 60 programming language. The two major analysis and design techniques, Structured Analysis & Structured Design (SASD) and object oriented analysis and design (OOAD) can trace their roots back to about the same date when both the class and structured programming were introduced as concepts [5].

SASD and OOAD diverged almost immediately thereafter, with structured programming becoming very popular from the late 1960s, through the 1970s and early 1980s and finding its way into many large scale commercial software projects. For much of this period OOP remained in research labs and educational institutions and there were few languages that supported the concepts. Smalltalk was created during the 1970s [6], C++ [7] and Objective-C [8] in the early 1980s and Eiffel in 1985 [2]. Many other OO languages were created during this period but most have not survived in widespread use. Object oriented languages may be classified as either hybrid, building upon an already existing procedural language, or pure. Smalltalk and Eiffel are pure OO languages whilst Simula and C++ are hybrids.

Over the same period several OOA and OOD methodologies and notations have been created. A recent bibliography discussed 21 such notations [9]. Many share similarities, not surprising given their common support of the underlying object model. Perhaps the most popular are Booch [1] and Object Modelling Technique (OMT) [10].

1.6 OOP within HEP

The history of OOP within HEP goes back to a project using Simula in 1978 [11]. Other relatively early projects that I’m aware of are the Pions project at CERN [12] and the REASON project [13] at SLAC.

Despite these pioneering efforts, most projects involving OOP within HEP are still relatively small scale, and have tended to focus on components of on-line systems or graphics systems. However the Gismo [14] and GEANT4 [15] simulation frameworks and the work of several CERN R&D projects targeted at the LHC experiments are significant developments in the off-line environment. The BaBar collaboration at SLAC [16] is also intending to base its on-line and off-line code development on the OO methodology, using C++ as the implementation language. However, they recognise the large body of existing expertise in Fortran and C and are not excluding some continued development in those languages, albeit using object oriented analysis and design.

It is therefore clear that OOP is gradually becoming accepted within the mainstream of HEP software development. Crucial to this are not only the potential advantages of this technology, but also the continued ability to use the large body of legacy software, mainly written in Fortran. Unfortunately such inter-language interfaces are not well defined and this is an area of some concern. One possible solution to this problem is the Interface Definition Language (IDL) defined by the Object Management Group (OMG) as part of the Common Object Request Broker Architecture (CORBA) for distributed object applications [17]. This defines object interfaces in a programming language independent manner and also defines
bindings to several programming languages including C, C++ and Smalltalk. A prototype binding to Fortran 90 is underway [18].

1.7 Evolution or Revolution?

Given its common ancestry with structured programming, it is clear that OOP is evolutionary and not revolutionary in origin. Does that mean that the impact and potential results are only evolutionary and not revolutionary? The jury is still out on that issue. Reusability is a prime goal of OOP and this should bring about significant gains in productivity amortised across several projects. However, other than user interface and data structure libraries there have been relatively few successful class libraries in other problem domains. This might change with the work of the OMG and the distributed object services [19] component of CORBA. The real revolution will perhaps come when the focus of most people’s attention within the HEP computing community is on the complete software development cycle rather than the details of programming language issues. Analysis and design are hard and the OO approach considerably improves the intuitive decomposition of the problem and the flexibility of the resulting solution.

2 Case Study 1: On-line Run Control

The role of the data acquisition system in a high energy physics experiment is to take digitized event data from the front-end electronics, gather together the many fragments for a single trigger to form an event, perhaps reject events in a processor farm and then record those that survive on some form of archival storage.

A conceptual data acquisition system is shown in Figure 2a. It takes the form of a pipeline having several stages. The trigger indicates the occurrence of a potentially interesting physics interaction. Event fragments are digitized in the multiple front-end modules (FEM), processed in the read-out controllers (ROC), typically by having pedestals subtracted and channels with values below some threshold being suppressed. The Event Builder (EVB) creates complete events from the multiple fragments and the processor farm typically acts as a software trigger processor, performing a physics selection filter on complete events, accepted events being written to archival storage, typically magnetic tape. Events or event fragments might be buffered at some or all stages in the pipeline.

![Figure 2a: Conceptual Data Acquisition System](image)

![Figure 2b: Simplified State Transition Diagram](image)
The role of the Run Control application is to control the elements of this pipeline during the various stages of data acquisition - starting a data taking run, perhaps pausing and resuming it and finally ending it. It plays no part in the actual flow of event data from the detector through the data pipeline, but communicates with the pipeline elements via a (conceptually) separate control and monitoring network. Most elements in the pipeline will be programmable, operating under control of a network aware real-time operating system, and require both code and data to be downloaded to them, depending typically on the type of data taking run that is to be performed and perhaps the results of a prior calibration process. Because of the buffering at the various stages on the pipeline, care must be taken to ensure that all such buffers are correctly flushed when pausing or ending a run.

The following discussion is based loosely on the CODA Run Control application [20] for the CEBAF data acquisition system which was implemented in the Eiffel programming language [2].

2.1 The Abstractions

2.1.1 Finite State Machines

The main abstraction is that of a Finite State Machine (FSM). A FSM can belong in one of several states, responding to external stimuli by performing an action and transitioning to another state. For our example, all elements of the pipeline react to the same set of external stimuli, obeying the same State Transition Diagram as shown in Figure 2b. This describes the various phases of a data taking run and the transitions that take place. Some of the main transitions are:

- Configure. This transition determines the conditions under which the forthcoming data taking run is to be performed. It might involve selecting one of several run types (e.g. physics, cosmics, calibration) that will modify the details of subsequent transitions.
- Download. This transition involves loading the distributed elements with the information that is appropriate for the forthcoming run. This might be implemented as actually downloading all the code and data, or might involve just informing the elements of the configuration details, letting them upload themselves over the network.
- Prestart. This primes the system for a new run without performing a new download operation, perhaps zeroing out scalers etc.
- Pause & Go. These initiate and halt data taking temporarily during a run.
- End. This terminates a run, perhaps closing the file on the output device.

2.1.2 Pipeline, Pipeline Stages and Pipeline Elements

The conceptual data acquisition system is based on a pipeline having multiple stages, each stage having possibly multiple elements. Each element is accessible from the host computer via the control and monitoring network and this access forms the basic granularity of the system. Typically the FEMs are not intelligent enough to be directly attached to the network so the ROCs are treated as agents for communication to them. If this simplified model is inadequate, the FEMs are treated as the second pipeline stage, the ROCs as the third. The event builder, processor farm and output device form the remainder of the pipeline stages.

A Pipeline Element is a software object that acts a proxy for the corresponding hardware element in the pipeline. The details of the communication protocol can be deferred to a discussion of the implementation classes. It will not be discussed further here.
The pipeline, the pipeline stages and pipeline elements all act as FSMs, obeying the same state diagram. This view raises the issue as to what state the pipeline or a pipeline stage is considered to be in while it is waiting for all its elements to complete their transitions. In the formal FSM model each transition is considered to be instantaneous, but in our model, it might take a finite amount of time for an element to complete a transition. This can be addressed by introducing additional “transitioning” states such as “going”, “pausing”, “downloading” etc.

Finally one must address the issue of how to deal with a situation where a transition is initiated but for some reason one of the elements fails to complete it successfully. The approach taken for this example does not conform to the formal FSM approach, but assumes that the pipeline as a whole will also fail to make the transition and so remain in its original state, albeit indicating that an error has occurred. All elements apart from the element that failed will however have completed their transition to the new state. Once operator or automatic intervention has corrected the problem with the failing element, the pipeline can be re-transitioned. This will act as a no-operation on all but the failing element since they are all in the desired state; only that element will be re-transitioned to re-synchronize the pipeline.

2.1.3 Sequencing

One aspect of the problem that is not immediately obvious is that the sequencing by which the various pipeline elements are transitioned. In general the elements at the same pipeline stage are independent of each other and so may be transitioned in parallel, but the different pipeline stages will need to be transitioned sequentially. Consider the situation when issuing a “Go” request to activate data taking. It is essential that the request is first issued to the tail of the pipeline, then to the penultimate stage and so on until the hardware trigger at the head of the pipeline is enabled. Conversely, when pausing a run, the pause request must first be issued to the hardware trigger, then the system must wait for event fragments to be flushed out from the next stage before the subsequent stages can be paused in sequence.

A simple sequencer might just be hardcoded to loop over the pipeline stages in the desired direction for each transition, but a more complex one might allow more flexibility.

2.1.4 Transitioners

At any state in the State Diagram, only a few stimuli and transitions to new states are valid. For example, it is not possible to transition from the Booted state directly to the Prestarted state. These restrictions are handled by the concept of transitioners that act on a target object, taking it from an initial state to a final state. The target of a transitioner could be the complete pipeline, a pipeline stage or an individual pipeline element. Transitioners understand the intermediate transitioning states and apply timeouts in order that the pipeline should not stall if a single element fails.

One possibility would be to allow sequences of allowed transitions. Thus one might be allowed to transition directly from the Configured to the Active state by passing through the Downloaded and Prestarted states. This would allow flexibility in the user interface and perhaps speed error recovery operations.

2.1.5 Data Taking Run

This concept just embodies the overall behaviour of a data taking run. It is a slightly broader concept than the pipeline since it has additional information associated with it. For example, it holds the knowledge of the current run number and the run type. It has a containment association
with the pipeline. The data taking run is the abstraction that the operator really wishes to interact with when controlling the experiment.

2.2 The Classes

The class diagram is shown in Figure 3. This uses the Booch notation [1] where the arrowed lines denote an inheritance relationship (the arrow pointing towards the ancestor) and the lines with a solid ball at one end denote containment relationships, the cardinality of which is expressed by the numbers appearing alongside.

The STATE_MACHINE class imposes the Finite State Machine on its descendant classes. Thus it requires that they respond to “Configure”, “Download”, “Pause” etc., without specifying an implementation. The PIPELINE, PIPELINE_STAGE and PIPELINE_ELEMENT classes are direct descendants of STATE_MACHINE. The PIPELINE class contains an ordered list of PIPELINE_STAGE objects and the PIPELINE_STAGE class contains a list of PIPELINE_ELEMENT objects. In this simple implementation the sequencing of transitions is performed by appropriate traversal of the list of PIPELINE_STAGE objects contained by the PIPELINE object. One difference between the implementations of the PIPELINE and PIPELINE_STAGE classes is that the PIPELINE class must transition its pipeline stages sequentially, whereas the PIPELINE_STAGE class transitions all its PIPELINE_ELEMENTS in parallel.

The RPC_ELEMENT and CORBA_ELEMENT classes are concrete classes that implement specific communication protocols between the proxy objects and the corresponding hardware.
The $\text{DAQ\_RUN}$ class adds the attributes of the run number and run type and contains a single instance of the $\text{PIPELINE}$ class. It performs transition or status requests by delegation to the pipeline object.

The diagram is somewhat simplified in that not all the relationships between the $\text{STATE\_MACHINE}$ hierarchy and the $\text{TRANSITIONER}$ class are displayed. Similarly, descendant classes of $\text{TRANSITIONER}$ corresponding to the various transitions are not displayed.

### 2.3 Eiffel Language Support for OOP Concepts

This application is implemented in the Eiffel programming language. Its support for the various OOP concepts discussed in Section 1 is shown in Table 1.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Implementation</th>
<th>Code Example</th>
</tr>
</thead>
</table>
| Abstraction              | Deferred Classes                 | `deferred class FS_MACHINE
                        ....
                        pause ( run: DAQ\_RUN ) is
                        deferred
                        end;`                                           |
| Encapsulation            | Explicit export control          | `class DAQ\_RUN
                        feature (NONE)
                        ....
                        feature (DAQ\_RUN)
                        ....`                                          |
|                          | Restricted export                |                                                  |
| Inheritance              | Single & Multiple                | `class RUN inherit
                        FS\_MACHINE;
                        feature
                        ....
                        end`                                           |
| Polymorphism             | Unless $\text{frozen}$           | `class RUN
                        feature
                        frozen set\_name( name: STRING ) is
                        ....
                        end;`                                          |
| Genericity               | Generic Classes                  | `class LIST[T]
                        class HASH\_TABLE[T->HASHABLE]`                  |
|                          | Constrained Genericity           |                                                  |
| Lifecycle                | Object Creation                  | `class RUN
                        creation
                        make
                        !!run.make;`                                    |
|                          | Garbage Collection               |                                                  |

Table I: Eiffel Language Support for OOP Concepts

### 3 Case Study 2: An Application Framework

The described application framework is suitable for general HEP reconstruction programs and is designed to provide flexibility without the need for recompilation. The implementation upon which this discussion is based is described in Reference [21].
Underlying concepts for the following discussion are those of data taking runs and events. A data taking run, identified by a run number, is a management unit of stability during data acquisition for an experiment. Such runs are characterized by having sets of adjustable parameters that are appropriate for data accumulated within the run, but which might differ from the values for adjacent runs. Thus any analysis code that wishes to operate on the data for a particular run must be given the opportunity to first access the appropriate parameters.

An event comprises the data for a single interaction or trigger of the experimental apparatus and is typically organized as a hierarchical set of data structures corresponding to the digitized output from the various detector subsystems.

A typical scenario for event reconstruction is the following: the events from one or more data taking runs are used as the input to an application that manipulates them, perhaps subsequently rejecting some of them as being uninteresting physics, or sorting them according to the underlying physics processes.

### 3.1 The Abstractions

#### 3.1.1 Modules

The application framework is based on the concept of **modules**. A module is a fragment of executable code that has a well-defined interface and performs a well-defined service. The interface is imposed by requiring that each module inherit from an abstract parent class. Generally modules are totally independent of each other, operating purely on the basis of their own internal configuration, data taking run specific information and the input event data. A module might generate new information which might be added to the existing event information or might perform a filter function based on the event characteristics or might perform some statistical operation, integrating the results from multiple events.

Each module will provide an interface to the framework that includes a unique name and functions that will be called at the beginning and end of the job, at the beginning and end of each data taking run (i.e. when the run number changes) and a per event function.

Several types of specialized modules are supported within the framework. These include the following:

- **Input modules**, which act as the source of data. One example might read data from a data file or files, another might access events from the event server in the on-line environment and another might select events on the basis of a collection of objects stored in an object-oriented database (OODBMS). In all of these cases the remainder of the executable modules should be unaffected by the origin of the data. Only one Input Module can be active at any one time.

- **Output Modules**. These act as the sink of data. One will output event data to several possible output data files, thus supporting the concept of simultaneous output streams. Another will make events available to the on-line event server and another might store updated events in an OODBMS.

- **Filter Modules**. A filter module can terminate or re-direct the subsequent processing of an event based on its filter criteria and the characteristics of the event. A simple filter module can signify success or failure, terminating processing of the event in the latter case. A more complex filter module can act as a switcher, causing subsequent processing to be redirected to one of several other modules.
3.1.2 Sequences and Paths

Multiple modules can be combined into a sequence having a unique name. A sequence may also include other sequences to provide an arbitrary nesting depth.

A path is a list of modules and sequences that begins at the input module and terminates at the output module. The processing for a path may be prematurely terminated by the action of a filter module. Multiple paths are supported, corresponding perhaps to different physics processes.

3.2 The Classes

The framework class diagram is shown in Figure 4. The Sequence class appears twice because a sequence can itself contain another sequence.

3.2.1 Executable

The basic underlying abstraction is that of an Executable, an entity that provides the following services:

- A beginning of job service. This is requested once per job.
- A beginning of run service. This is requested whenever the run number changes.
- A per-event service. This is requested for each event that enters the processing chain.
- An end of run service. This is the complement of the beginning of run service.
- An end of job service. This is requested at the end of each job.

Note that these services are purely abstract, the main goal of the Executable class being to impose the interface upon its children, forcing them to provide implementations.
3.2.2 Module

The interface for the Module class is identical to that of Executable, but here it makes sense for there to be some default action for each of the services. This default implementation is a null operation (i.e. no action), but this is different from the situation with the Executable class, where there was no implementation specified. Each instance of the Module class may be enabled or disabled. This allows an application to be created containing a large set of modules, only some of which are of interest at any one time.

User supplied modules will inherit from Module, overriding the default implementations as they require, providing specific implementations for them as appropriate.

TheInputModule, OutputModule and FilterModule classes are specialization of Module

3.2.3 Sequence

The Sequence class exhibits the identical interface to that of Executable, but the implementation of each service involves delegation to the members of the sequence. Thus the concrete implementation of each service will involve looping over all members of the sequence, delegating the request to them. In the case where the member is itself a sequence, it will again delegate the request to its members. This is an example of polymorphism, where several classes exhibit the same interface but have different implementations.

3.2.4 Paths

A Path is a specialized sequence and has the identical interface, and in the simplified model, the same implementation. However, it is treated as a descendant class because it has specific attributes in the full implementation that are not discussed here.

3.2.5 Framework

What is not perhaps immediately obvious is that the framework itself exhibits the same interface as Executable. It must provide services for the beginning of the job, beginning of run etc. It is only the implementation that is specific. In particular:

- The beginning of job service is delayed such that the beginning of job service for each module is accessed on the first occasion that the module is enabled.

- The beginning of run service is implemented as a loop over all enabled modules, accessing their beginning of run service.

- The end of run service is implemented as a similar loop to the beginning of run service.

- The end of job is implemented as a loop over all modules that have been enabled at some point in this execution, accessing their end of job service.

- The per-event service is implemented as an access to the event service for the enabled input module, followed by a loop over each path, accessing the event service for each sequence or module in the appropriate sequence. If a path is terminated through the action of a filter module, processing proceeds to the next path. Finally, the event service for the enabled output modules is accessed.
3.2.6 Lists and Hash Tables

Common to many of the application-specific abstractions is the concept of a list. A list is an object that chains other objects together, allowing a client to ask for each object in sequence. It has a head and a tail and mechanisms for inserting or removing elements.

Many possible implementation of a list are possible. The most space efficient are those based on arrays, but these are less efficient for adding or removing items within the list itself. Other implementations are singly (or doubly) linked lists, where node objects are defined that contain a pointer to the next (and previous) node as well as containing the item itself.

The main point here is that the application code should deal with an abstract list interface without worrying about the details of the implementation other than when instantiating the list. A subsequent decision to change the requirements resulting in a change in the list implementation (perhaps to allow for more efficient backwards traversal) then has very localized consequences; only the actual instantiation code has to be modified.

Another requirement is that the names of modules, sequences and paths be unique. Thus each new module must have its name checked against all the already existing ones. Whilst a list could be used for this purpose, a hash-table is a more efficient method of performing this, especially in this environment where wild-carding of names is not allowed. A hash-table uses a simple hash key, in this case derived from the name, and stores items indexed by this key, resolving collisions where two names hash to the same key. Access is then very efficient, most items being located following a single hashing operation. Contrast this to the situation where a list of names is maintained, every item of which has to be checked for a name clash.

The important point here is that the interface to an abstract hash table should be defined. Several different concrete classes that provide different implementations of this interface might then be provided. Initially an implementation based on a list class might be adequate. Eventually tests might show that the performance of this was no longer acceptable and another, more efficient concrete class having the same interface might be implemented. In this case the only change to the user code would be to instantiate the appropriate concrete class. No other changes would be necessary.

4 C++ Language Support for OOP Concepts

This framework is implemented in the C++ programming language. It’s support for the various OOP concepts discussed in Section 1 is shown in Table 2.

<table>
<thead>
<tr>
<th>Concept</th>
<th>Implementation</th>
<th>Code Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstraction</td>
<td>Pure Virtual Classes</td>
<td>class Executable {</td>
</tr>
<tr>
<td></td>
<td></td>
<td>....</td>
</tr>
<tr>
<td></td>
<td></td>
<td>virtual event( ) = 0;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td>Encapsulation</td>
<td>Public, Protected &amp;</td>
<td>class Executable {</td>
</tr>
<tr>
<td></td>
<td>Private Members</td>
<td>public:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>....</td>
</tr>
<tr>
<td></td>
<td></td>
<td>protected:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>....</td>
</tr>
</tbody>
</table>

Table II: C++ Language Support for OOP Concepts
5 Conclusions

Many of the abstractions in these case studies are intuitively obvious, this being one of the main attractions of the object oriented approach. Most of them were identified during the analysis phase of software development, but some were only identified later during the design phase. In particular, inheritance relationships are mainly identified during the design process. Inheritance is a powerful tool in both enforcement of a common interface and in providing a hierarchical specialisation of concepts. Polymorphism provides a complementary mechanism for simplification. These example demonstrate the support for OOP concepts by two different programming languages, one of which (C++) is a hybrid, being based on an existing procedural language, the other of which (Eiffel) is a pure object oriented language.

6 References


<table>
<thead>
<tr>
<th>Concept</th>
<th>Implementation</th>
<th>Code Example</th>
</tr>
</thead>
</table>
| Inheritance              | Single & Multiple           | `class Module : public Executable { 
....
};`                                                |
| Polymorphism             | Virtual Functions           | `class Module : public Executable { 
....
    virtual void event();
};`                                                   |
| Genericity               | Templated Classes           | `template<class Item>
    class List {
    }`                                                   |
| Lifecycle                | New & Delete                | `Path* aPath = new Path( "MyPath" );
....
    delete aPath;`                                      |
|                          | Constructor & Destructor    |                                                   |
|                          | No Garbage Collection       |                                                   |

Table II: C++ Language Support for OOP Concepts
1978.


Applying an Object Oriented Approach to Off-line Reconstruction at the LHC

A. (RD) Schaffer
LAL, Orsay, France

Abstract
Using an example taken from work on a prototype for electron track reconstruction in the Atlas detector, this paper will try to guide the reader through an object oriented (OO) design. It will attempt to illustrate how to approach a problem in an OO manner using a few different modelling techniques. In particular, we will introduce the use of patterns to describe object structure and behaviour.

1 Introduction

When one is used to programming in a procedural language, such as C or Fortran, and he/she first attempts to do an object-oriented development, it is often confusing as to where to begin. I do believe that it is important to learn to approach software development in a systematic way, for instance by using a global process development model as is described by the ESA standard [1], matched with a software methodology such as Booch [2] or OMT [3] for object-oriented developments. However, when first starting off, it can be difficult to see how ‘theory’ can be applied.

In the following, I will try to explain in a simple manner how one can attempt to solve a problem using an object-oriented approach. We will follow an example which is derived from work on a prototype* which attempts to reconstruct electrons in the Atlas detector.1 The steps we will follow can be summarized as:

• Understand the reconstruction algorithm
• Identify some initial categories of classes
• Describe some of the basic scenarios, i.e. jobs to be done
• Walk through the different scenarios to identify the classes needed and to decide ‘who does what’

One of the central activities of an object-oriented development is to identify the key abstractions of the problem. This is true whether one is trying to identify the different objects which are communicating with each other, or even to abstract different ‘patterns’ of object structure or behaviour. I highly recommend the interested reader to have a look an excellent discussion on design patterns [5] which has recently appeared. This book captures a large amount of experience in object-oriented design and can help boot-strap oneself by learning from others’ experience. In the following, I will use their patterns to help establish a standard design vocabulary, and will also adopt their diagram notation.

* This work is currently being carried out within the Moose Collaboration [4], a research project investigating the feasibility of adapting an Object Oriented approach to off-line software development for the LHC.
1 Atlas will be one of two experiments at the LHC at CERN exploring 17 TeV centre-of-mass energy pp interactions. It is expected to turn on in 2004-2005.
Overview of the problem: electron reconstruction

Electrons can be recognized by a characteristic signature in the detector:

- An electromagnetic shower in the calorimeter, and
- a track found in the Inner Detector which points to the shower cluster, and has a measured momentum which agrees with the energy seen in the calorimeter (E/P).

This can be seen schematically in Figures 1 and 2:

![Figure 1: Transverse view (ρ - z) of an electron signature](image1)

The electron trajectory is a straight line in this projection.

![Figure 2: End (ρ - φ) view of electron signature](image2)

An electron track is made of 6 space points which fit to a helical trajectory with deviations due to multiple scattering and Bremsstrahlung. Each space point is made from a hit in 2 adjacent layers of precision Si-strip detectors. The Inner Detector is composed of six cylindrical superlayers, each with a pair of Si-strip detectors:

![Figure 3: View of divisions of Si-strip detectors](image3)
A key point driving the tracking algorithm is the high luminosity at LHC:

- At each beam crossing there are on average 20 interactions.

This implies that an interesting physics event has 100’s of ‘background’ tracks. Thus the strategy to approach the problem is not to try and reconstruct all tracks, but to ‘seed’ the search with information from another detector. In the case of electrons tracks, we seed the reconstruction Inner Detector in with an electromagnetic cluster found in the calorimeter. This search path is called a road.

This leads to the following simplified procedure for track fitting:
1. Construct a wide road from a calorimeter seed
2. Collect Si_strip detectors which lie inside the road
3. Construct space points in outer superlayers (lower density of hits)
4. For all pairs of space points
   - fit with vertex and seed
   - construct a narrow road and
   - collect all hits in road
   - fit set of hits
   - select candidates satisfying certain criteria, e.g. lowest $\chi^2$, etc.

Thus, one is looking for the six space points in Figure 2 which in a real event will be superimposed with thousands of other hits from other tracks, noise, etc.

3 Identifying the classes and scenarios

From the description of the problem, it is fairly easy to come up with a list of candidates for objects:

- Detector objects: Inner detector, Si strip detectors, superlayers, Si rings, electromagnetic calorimeter
- Reconstruction objects: EM energy clusters, inner detector hits, space points, seeds, roads, tracks, electrons

What are some of the possible scenarios?:
1. Clearly, for each event we will need to ‘input’ the raw data produced by the different detectors, e.g. inner detector hits and calorimeter cells with energy.
2. There is initial processing of detector data, e.g. electronic calibration and initial clustering of data.
   As we begin the track reconstruction, we need to:
3. Generate roads from calorimeter clusters
4. Collect detectors and hits in road
5. Build space points from hits
6. Fit and iterate 4-6.

3.1 Dividing up the problem

The next step is to divide up the problem domain by identifying some of the basic categories, or groupings, of objects and defining their primary responsibilities. For the present problem, we can identify 3 categories:
• The ‘raw data’ and the subsequent reconstruction quantities which are to be saved should be grouped together as an event.

• We need to have a ‘model’ of the detector which is responsible for:
  • ‘knowing’ its geometry
  • accessing the raw data of each new event
  • perform the ‘initial’ steps in reconstructing its own data

• We will probably need to ‘invent’ some ‘reconstructor’ objects to help in the track finding

We conclude that we need event, detector, and tracking categories.

We will mostly concentrate on the detector model, where our underlying ‘design’ philosophy is to allow the detector to do ‘as much possible’ before inventing new objects to do the work.

4 Definition of notation

We will use three types of diagrams² to illustrate the design which we develop: Class diagrams, Object diagrams, and Interaction diagrams. Figure 4 shows the various relationship between classes:

![Generic Class diagram](image)

**Figure 4:** Generic Class diagram

Here subclasses which inherit from an abstract class is represented by a triangle. And a class can ‘refer’ to another class which is given by a straight line, terminated in either an simple arrow or arrow plus dark circle. This termination identifies the number of ‘destination’ objects referred to by each ‘source’ object.³ One can also indicate the functionality of an operation with pseudocode.

Class diagrams represent a static vision of classes. However, when a program runs it is the instances of classes, often just called objects, which are created and interact. It is often useful to

---

² This notation is taken from reference [5].

³ For example, a one to one relationship means an Abstract_class object will hold a reference to one Concrete_class object, and a one to many relationship means an Abstract_class object will hold a reference to a List or Set of several Concrete_class objects.
give an example of the object structure at a certain moment during a program run. This is expressed in an object diagram:

![Diagram](image.png)

**Figure 5:** Generic Object diagram

Here the rounded box refers to an object, and the reference from one object to another is given by an arrow. Note that the explicit name of the object reference is not always shown.

The last diagram, an interaction diagram, describes how a group of objects carry out a scenario. This diagram will be presented and explained later in an example.

### 5 The detector model

The detector model is a natural place to introduce a parent/child hierarchy or tree structure.

For example, we would like to have some structure of objects such as:

![Diagram](image.png)

**Figure 6:** Detector object diagram

In words, one would describe this as an atlas object contains an em calorimeter object and an inner detector object. Similarly, the inner detector contains three superlayer objects, etc. Each of these objects would probably be instances of a Detector class.
5.1 The Composite pattern

This detector hierarchy can be well described by the Composite pattern:

This pattern combines into the single interface of Component all of the operations that a client needs to access for both containers and primitives. One can ‘read’ this diagram as follows: a client refers to a single instance of Component. This component can either be a Composite or a Leaf object. A Composite object refers to other Component objects through the children reference. And again, these components are either Composites or Leaves. Thus when a client make requests of a component, if it is a Leaf, requests are handled directly. If it is a Composite, requests are forwarded to its children, with Composite possibly performing additional operations.

So how would this look like for our detector hierarchy?:

---

4 This structure describes a tree which is composed of nodes where each node is either terminal (leaf) or not (composite).
Referring back to Figure 6, one sees that the Leaf objects are the EM_calorimeter and Si_strip_detectors, and that the rest are Composites. Note that when assigning operations to the different classes:

- default operations are put into Detector_Component, Detector_Composite and Detector_Leaf.
- and these are over-ridden by the subclasses where specialization is needed.

6 Assigning responsibilities to the detector hierarchy

Walking through scenarios is a useful way of identifying the various operations that are needed and who is responsible for performing them. This further clarifies the roles of the different objects and is useful to uncover new objects/classes that may be needed.

6.1 Creating the detector hierarchy

The first job to be done is to create and initialize our detector hierarchy. We define an initialize operation which will be abstract in Detector_Component and implemented in Detector_Leaf and Detector_Composite. The initialize operation simply reads a geometry description file of the form:

```
atlas                  {    class { Atlas }
    children { inner_detector, em_calorimeter }              }
em_calorimeter {   class { EM_calorimeter }                                           }
inner_detector   {   class { Inner_detector }
    children { superlayer1, superlayer2, superlayer3 }    }
```

and, depending upon what is in the file, it will initialize the geometry of each object, create the children objects and continue the initialization recursively. This can be described as:

```
operation active on object

an_external_client     a_parent : Atlas     a_child : Inner_Detector
new Atlas
initialize(geom_file)
new Inner_detector
initialize(geom_file)
```

The parent child initialization continues recursively until the end of the geometry description.

Figure 9: The Initialize interaction diagram

---

5 An abstract operation is one whose calling signature is defined in the superclass and whose body is implemented in subclasses.
This diagram presents a sequence of messages between objects where each object is represented by a vertical line and the duration of an operation is shown as a box. The arrows between objects are the ‘messages’ or operation invocations, where creation is indicated by a dashed arrow.

Figure 9 presents only the first few top level interactions showing how an Atlas object is created, initialized and continues to initialize the rest of the hierarchy, driven by the description in the geom_file. Note that an external client is required to initiate the creation of the detector hierarchy. We leave this client unidentified for the moment.

6.2 Accessing detector data

With the detector hierarchy in place, we can look back at our first scenario:

1. ‘input’ the raw data produced by the different detectors

The first question we must ask ourselves is ‘who has data?’. In our simple model only the Detector_Leaf subclasses have data, that is the EM_calorimeter has various cells with energy and the Si_strip_detectors have position measurements or hits:

![Diagram](image1)

**Figure 10:** Class diagram of detectors with data

We need an ‘event structure’ to hold the raw data and eventually the results of the reconstruction. This we represent symbolically as an event object with ‘raw data attached to it’:

![Diagram](image2)

**Figure 11:** Simplified view of event raw data

So how does the detector hierarchy access its raw data? This can be done in a similar way to the *initialize* operation with a *trigger*(event) operation which pass an event through out the detector hierarchy:

- default operations in Detector_Leaf and Detector_Composite either do nothing or simply pass the event to the children.
- these operations are over-ridden in EM_calorimeter and Si_strip_detector. These specialized trigger operations must know how to ‘navigate’ from *event* to *raw_data* and extract their corresponding data.

An object interaction diagram for *trigger*(event) would be similar to Figure 9, where again an external client is responsible for looping over events, sending each one to the hierarchy.
7 Beginning the reconstruction

So who starts things off? Let’s give atlas a *reconstruct* operation who first job, scenario 2, is to request all detectors in the hierarchy to ‘*preprocess*’ their data (i.e. electronic calibration, initial cluster building, etc.):

![Interaction Diagram](image1.png)

*Figure 12:* The *reconstruct* interaction diagram

Now recall the next two steps in the track reconstruction:

3. Generate roads from calorimeter clusters
4. Collect detectors and hits which are inside a road

We clearly need a Road class, and we could leave the responsibility for the rest of the track finding, from step 4 onwards, up to the inner detector:

![Continuation Diagram](image2.png)

*Figure 13:* Continuation of Atlas *reconstruct* operation

7.1 Collecting detectors and hits

We want inner detector to use a road to collect the subset of tracking detectors and their corresponding hits which fall inside the road. There are the following constraints on this subset:

- Want to be able to select further subsets while looking for track candidates, and
- Would like to preserve the ‘structuring’ of the hits which is provided by the detector hierarchy.
The solution we have chosen is to construct a **Tree** of **Node** objects, where each node refers (points) to a selected detector component or hit:

![Class diagram of a Tree](image)

**Figure 14:** Class diagram of a Tree

Again we encounter our Composite pattern, where we have a hierarchy of nodes with a Tree object pointing to the root node. Here Any_object can be either a Detector_Component or a Si_hit.

In order to construct the set of nodes and the tree, inner_detector will:

1. create a tree and a root node
2. attach itself as an Any_object, and
3. pass the road and root to it’s children through a build_tree method

![Build tree interaction diagram](image)

**Figure 15:** Build tree interaction diagram
After sending the build_tree method through the detector hierarchy, we end up with the following set of objects:

Figure 16: Object diagram of tree with detector components and hits attached to nodes

Figure 16 gives a typical snapshot of the collected detector components inside the road. The arrows pointing to ... in the figure indicate that there are nodes, si_strip_detectors and si_hits which have not been drawn.

8 The Visitor pattern

Now before we begin tracking with our tree, let’s review the interface of Detector_Component:

<table>
<thead>
<tr>
<th>Detector_Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>initialize(geom_file)</td>
</tr>
<tr>
<td>trigger(an_event)</td>
</tr>
<tr>
<td>preprocess_data()</td>
</tr>
<tr>
<td>build_tree(road, node)</td>
</tr>
</tbody>
</table>

Figure 17: Class interface of Detector_Component

At this point, we can worry a bit about evolution of our model:

What is most likely to change in the future:

a) the Detector_Component hierarchy, or

b) the operations to be performed on the hierarchy?

Since the detector hierarchy closely resembles the hardware structure of the detector, which evolves fairly slowly, I would expect the operations to be more variable.
In this case, we can employ the VISITOR pattern:

We replace the four operations of Detector_Component by a single accept operation which takes as an argument a Visitor object:

\[ \text{Detector\_Component} \]

\begin{itemize}
  \item initialize()
  \item trigger(an\_event)
  \item preprocess\_data()
  \item build\_tree( road, node )
\end{itemize}

\[ \text{Detector\_Component} \]

\[ \text{accept( Visitor v ) } \]

\[ v->\text{visit\_component( this )} \]

**Figure 18:** Simplification of the Detector\_Component interface

In Figure 18, the pseudocode indicates that the **Visitor**, v, is simply requested to **visit\_component** and a reference to the current object, **this**, is passed as argument. This is what is called a **call-back** mechanism.\(^6\) Note that **visit\_component** is a **specific** operation for the Detector\_Component class.

The functionality of the removed operations is captured in different Visitor classes:

**Abstract\_Detector\_Visitor**

\begin{itemize}
  \item visit\_component( Detector\_component )
  \item visit\_superlayer( Superlayer )
  \item visit\_si\_strip\_detector( Si\_strip\_detector )
\end{itemize}

**Initialize\_Visitor**

\begin{itemize}
  \item visit\_component(Detector\_component )
  \item visit\_superlayer( Superlayer )
  \item visit\_si\_strip\_detector( Si\_strip\_detector )
\end{itemize}

**Trigger\_Visitor**

\begin{itemize}
  \item visit\_component( Detector\_component )
  \item visit\_superlayer( Superlayer )
  \item visit\_si\_strip\_detector( Si\_strip\_detector )
\end{itemize}

**Superlayer**

\[ \text{accept( Abstract\_Detector\_Visitor v ) } \]

\[ v->\text{visit\_superlayer( this )} \]

**Figure 19:** Class diagram of detector visitors and superlayer

With visitors the functionality of an operation, for example initialization, is grouped together into a single class. And each visitor must provide a specific operation for each of the classes which will **call-back** to the visitor. Of course default operations can be provided, e.g. in Abstract\_Detector\_Visitor. This pattern allows the adding or the changing of operations in an easy way, since the Detector\_Component classes can accept any visitor which inherits from Abstract\_Detector\_Visitor. But as a consequence, when adding a new type of Detector\_Component, each of the visitors must add a new method.

---

\(^6\) For the more advanced readers, this mechanism is called **double-dispatch** where the **operation** which is executed depends **not only** on the type of the object receiving the request, but also on the type of the object sending it.
9 Track finding, cont.

We will now introduce a few more classes to schematically outline how the rest of the track finding will proceed. This will lead us to introduce our final pattern: Strategy.

After the detectors and hits have been collected in a tree, we allow Inner_detector to delegate to a Track_finder object to extract hits from the tree and generate the various Track_candidates:

Figure 20: Track finder extracts hits from a tree to create track candidates

Clearly, Inner_detector will have to pass to the Track_finder the Tree and the Road. We can then implement the Visitor pattern for our tree hierarchy so that a set of Tracking_tree_visitors can be used to collect the hits, etc. The overall pattern recognition algorithm is implemented in the methods of our Track_finder.

9.1 Introducing the Strategy pattern

Suppose that the Track_finder wants to request its candidates to fit their hits with different fit methods. How does one avoid putting the multiple fit methods into the Track_candidate’s interface? One solution is the strategy pattern:

Figure 21: Track_candidate using a Fitter strategy

The Strategy pattern allows one to vary transparently the way something is done. In the present situation, the Track_finder may want to begin with a simple fit algorithm and to progress to a more sophisticated one. Thus Track_finder creates the fitter of interest and passes it to the Track_candidate through the set_fitter(Abstract_Fitter fitter) operation. When Track_finder later requests Track_candidate to fit(), the Track_candidate will use its reference to fitter. As can be seen in the pseudocode of the figure, the fitter is passed a reference to the current
Track_candidate, \texttt{fitter->init\_fit(this)}, which allows the fitter to access, i.e. call-back, the Track_candidate to get the needed information for fitting.

The strategy pattern shows how one can cleanly encapsulate a variation: a Track_candidate only knows that it has some type of Abstract_Fitter, and the true fitter is determined by the client who is controlling the fitting action of the Track_candidate.

10 Summary and conclusions

Learning to design software from an object-oriented point of view can be lots of fun, although it may not be so easy to learn at first (at least for the ‘older’ generation).

A simple procedure which can help is:
1. State the problem
2. Generate a number of scenarios of things to be done
3. Walk through the scenarios to identify the participating classes and their responsibilities.
4. Use standard patterns, wherever possible, for object structures and behaviour.

It is extremely important to discuss with others while designing and to have others review your work.

References

First Driving Classes on the Information Superhighway

Wulf Bauerfeld
DeTeBerkom, Berlin, Germany

1 Is the Information Superhighway necessary?

Recently, some statements were published on the unnecessity of the Information Superhighway. According to the evaluation of a marketing company any effort on implementation of information highways is meaningless since:

- in the mid-80s, the EC funded a number of broad band projects (RACE); none of these trials have matured yet into commercial applications.
- the concept of a broad band ISDN is nearly 10 years old. Standards have been in place since 1990 and the technology has yet to find a commercial application or be implemented.
- the massive growth in the use of consumer on-line data-bases such as Prodigy and America OnLine has been supported largely by X.25, widely held to be an outdated, narrow band technology. But when Sprint lowered its X.25 tariff, network traffic tripled.
- X.25 is being superseded by frame relay in the business environment and the latter seems to provide sufficient bandwidth for the majority of business applications. In an annual survey only one of 1000 managers of corporate networks interviewed said that lack of capacity was the greatest problem he faced in operating his company's network.
- nearly two-thirds of existing cable capacity on transatlantic and transpacific routes is unused according to a study commissioned by COMSAT (US satellite operator).
- "Video-on Demand" which has been heralded as the main consumer application for broad-band networks, is not expected to generate sufficient revenues to fund a national roll out in any market.

These observations are certainly true but lack the notion of multimedia" data, i.e. access, retrieval and representation of digital data which is far beyond the classical alphanumerical input/output of today's commercial application.

"Multimedia is information that comprises basic and application-specific information types. It is intended for visual and auditory perception as well as for automated processing. This information is not restricted to local existence, but may have be distributed. The information may, but does not have to be, presented on different presentation media."

With the potential of this upcoming technology in mind the buzzword "Information Superhighway" cannot be neglected, implementations problems and technical issues will be, therefore, discussed in the following.

2. What is the Information Superhighway and what is it for?

The Information Superhighway has not to be implemented from scratch nor does its creation cause a "Big Bang". It will be an amalgam of (partially already existing):

- information networks,
- services,
- endsystems such as PCs or TV-sets
and will consist of thousands of interconnected, interoperable communication networks, including

- terrestrial and satellite broadcasting,
- cable television networks,
- wired and wireless telephone systems,
- commercial computer network services,
- the Internet and its successor.

The Information Superhighway has been described as "a system to deliver to all (Americans) the information they need when they want it and where they want it - at an affordable price." (Michael Nelson, OSTP). Development in three areas is, therefore, required to make this system working:

- interoperable, extensible endsystems so that television and communications can evolve along the technology curve;
- computer systems, televisions, telephones, and other devices converged to serve as "information appliances".
- digital libraries, information services, and databases to provide the information content;

Since potential users will buy content, not technology, content and useful applications will attract the usage necessary for success. Therefore, cultural industries, particularly motion pictures and television, must be seen as an essential part of the implementation strategy path which, nevertheless, means that intellectual property rights associated with cultural items are to be protected by audit and control mechanisms.

The Information Superhighway should be open to all information suppliers on an equitable basis, so that applications may include distribution of entertainment programming, educational information, government data, manufacturing information, and access to health care. A summarising, but not exhaustive list of applications shows some of the inherent advantages of this global system:

- digital program delivery over interactive networks will permit additional cost-effective services such as "shopping at home" and "multimedia information on-demand".
- electronic distribution of educational material will expose all students, even those in remote locations, to a high-quality education.
- electronic availability will facilitate access by business and the public to government reports, weather information, and other scientific data.
- electronic "blueprints" and rapid multimedia communication between manufacturers and their suppliers will improve manufacturing productivity.
- health care will be improved by telemedicine and maintenance of on-line medical records.
- telemedicine will enable remote patient examination and diagnosis.

Most probably the Information Superhighway will be built, owned, and run by the private sector with hundreds or thousands of companies providing services whereby the role of regulating bodies (such as the governments) is to ensure that these systems and services are interconnected and interoperable in order to provide competition and choice for the customer.
Since the goal is to have a fully competitive marketplace in which any company may provide any service to any customer we are witnessing a fundamental paradigm change: On the marketplace of the future we replace atoms by bits...and, consequently, material information transport by electronical information transfer.

The Information Superhighway will cause convergence of different players which found - up to now - their undisturbed and independent role in

- communications
- computing
- consumer products
- content provision

and will, therefore, be the catalyst of new alliances

For the time being its realisation is somewhat deadlocked since the Information Superhighway requires

- a network to develop new applications;
- applications to generate user demand;
- user demand to justify the network.

But there are several political ("Al Gore" Initiative, "Bangemann" Report, G7 Summit) and technical (ACTS - National Hosts, European ATM Trial - Telekom, Advanced Research Networks - SuperJANET, DFN, etc.) programmes discussed or already launched respectively.

3. Communication in Time and Space

Today's communication systems are based either on analogue or on digital technology and can be distinguished into circuit-switching (e.g. telephone), distributive (e.g. cable TV), or packet (cell)-switching (e.g. X.25) networks. The digital technology is about to replace the analogue technology in the telephone network (ISDN) and soon cable TV providers will launch digital television channels (following e.g. a Standard called MPEG).

Whereas analogue networks offer a channel with a certain bandwidth to the user (e.g. telephone = 3 kHz, stereo radio = 2 * 15 kHz, colour TV (PAL) = (MHz), digital networks provide a channel with a certain access or transfer rate given in bit per second. This rate might vary from 9.6 Kb/s for "normal" data connection to 10 Mb/s on an Ethernet, up to 155 Mb/s for modern "High-speed Networks". Note that the term "high speed network" is somewhat misleading, the bits are not faster than in a low speed network but shorter. A bit "measures" more than 30 km with 9.6 Kb/s, less than 2 m for 155 Mb/s which for the latter requires high speed electronics, of course.

Partners performing analogue information exchange are usually making continuous use of the information channel provided: even silence in a telephone conversation or the test-image in TV are converted into a continuous analogue signal and transmitted over the circuits. Since not everybody is telephoning with somebody else continuously the number of circuits between the switches can be less than the numbers of telephone subscribers: they share the circuits.
Digitally encoded information might be exchanged continuously but offers the chance of grouping bits into cells or packets and sending these cells/packets or ... nothing. This can happen either periodically or in a "bursty" mode.

Consequently, the basic idea behind the development of packet-switching networks was to offer the resources needed for providing a digital channel to other users during the time nothing is sent by the first user. Not circuits can be shared or multiplexed in digital systems with periodical or bursty input characteristics but links and storage in switching systems.

Of course, the statistical interlink in time with other users will influence under certain circumstances the data flow in the network. It might even change the input time pattern to an output pattern which is slightly or evidently different. In general, a behaviour of the "Transport Network" can be described as being

- isochronous
- synchronous
- asynchronous.

Isochronous behaviour is observed, if one does not need to go into the exact times of input and output time, thus it results from a ignorance of an always existing delay (since Einstein it is well known that signal transfer time is finite).

Synchronous behaviour is observed if one does not need to go into the details how much a delay may vary (jitter) which anyhow can be hidden to a certain extent by an buffering endsystem in which presentation of information is decoupled in time from its arrival in the system.

Asynchronous behaviour is from a certain level of granularity on the reality of any digital cell or packet switching system. Since resources are shared statistically in time with other users and these users' characteristics are undeterministic on bit/cell/packet level. Likewise internal error correction by repetition of packets, cells or bits changes the arrival of data in time at the endsystem.

Human senses have a characteristic time resolution of data (e.g. 16 slightly different pictures per second are enough to produce the impression of continuous moves). Thus certain time behaviour not in all cases observable or necessarily annoying. For instance, asynchronous behaviour becomes observable only, if resulting average delay > packet interarrival time. Since most computer-based applications behave anyhow bursty and, for instance a display of a personal computer can retain a pixel matrix as long as needed since its refreshing cycle is - in contrary to the TV-set - independent from the output of the network, a
asynchronously behaving high speed transport network is not disrupting drastically the multimedia sequences on the PCs.

4. "Asynchronicity" beyond the human senses

Specifically in fibre networks the asynchronous behaviour stems usually from the fact that links and switches are shared with other users - an economical way to offer information access to "an affordable price". Therefore, ATM (Asynchronous Transfer Mode) will be the principle new technology of the Information Superhighway. It also will comprise the typical existing asynchronous "Internet"-Technologies, i.e. packet transfer on serial links and on bus-structured Ethernets (CSMA/CD - Carrier Sense Multiple Access / Collision Detection) depicted below.

As long as the timing characteristics of these underlying networks are beyond the time resolution of the individual human senses, new digital services can be provided which allow the convergence of TV-based entertainment, interpersonal communication and multimedia information retrieval to an over the Information Superhighway.

Basically the difference between packets and cells is it size. In both cases information (either message or packets) are chopped up into smaller units (either packets or cells). Typical packet sizes are 1 - 2 Byte, typical cell sizes are 4 to 256 Byte (ATM: 5 Byte header + 48 Byte data). The smaller cells are filled quicker and can be forwarded earlier.
small packet caught behind big packet

serialization with cells

It can be derived easily that cells as the smaller units cause lower values in the time parameters of a asynchronous system, its "asynchronicity", therefore, might become less easily observable for human senses.

5. Why has cell networking suddenly become so important

The international association of telecommunication providers CCITT / ITU has decided to pursue a cell architecture for future telephone networks because it:
– reduces the number of transmission networks
– provides easier support for multicasting
– offers a better multiplexing scheme than (circuit switched) ISDN for high speeds

All these ideas are based on a famous and influential paper by J. Turner (1986) in which a detailed reasoning is given:

1) each communications service (telephony, CATV, computer network) has its own delivery network but will carry (digital) information encoded in bits; combining these separate networks into one has clear advantages such as
– simplified wiring
– economics of scale in network management
– easier integration of services

2) although circuit switched networking can support multicasting, it is easier to use packetized services like cells because such networks enable
– services such as digital (cable) television
– simple implementation of multi-party conferencing
– economic usage of links and connections

3) the standard ISDN model of providing a limited number of fixed-bandwidth circuits does not scale well because the original circuit bandwidth of 64 Kb/s chosen today is too much for voice (advances in encoding techniques) but still to little for video. Assuming a "Gigabit application" it has to set-up 16000 channels. Even if 16000 channels have been found in parallel the application might find it difficult to use them effectively.
6. Some reflections on service provision

The major time consuming resources in a network are the switches or routers. The challenge is to find a routing scheme which, both

- is fast because packets/cells must be routed quickly at high speeds
- requires a minimum of routing information in each packet/cell.

"Source-Routing" has been found fast and efficient for routing connection-less datagrams through a network. Each packet/cell contains a full address which most significant part is evaluated at each switch and then omitted comparable with address on letters sent with ordinary mail. But "source routing" has an inherent disadvantage: number of hops must be limited to ensure that the address header does not get too large.

If on the other hand the number of endsystems to addressed is large (e.g. every human being should be addressable) one ends up anyhow with 64 Byte headers as in IPn(ext)G(eneration) packets.

"Hop-by-hop Routing" requires a call-set-up information from end-to-end which generates entries in switching tables managed by the shops (switches or routers) according to the path between the endsystems with dynamically assigned numbers. After the communication has been disconnected these numbers can assigned to other "virtual" paths or channels.
For ATM, for instance, a two level hierarchy of hop-ids is foreseen. Nevertheless, these ids are much shorter than and independent of the length of the full endsystem identification.

Call or connection set-up of a connection-oriented architecture has another advantage which becomes rather obvious in asynchronous systems. Since the asynchronicity of the network is strongly depending on the actual number of users served and their statistical behaviour it would be nice to monitor and to control the number of users in order to provide a network service with non-disturbing time parameters. A connection-less architecture has to store-and-forward all cells which are put into the system. This might deteriorate the network behaviour for all users.

7. Connections are not the only problem

If, as a final conclusion, the Information Superhighway will be based on a connection-less (like in the Internet) or a connection-oriented (like in X2.45 or ATM) architecture remains open for the time being and requires more experience and understanding of critical network management issues. Moreover, a number of undesirable features of the Internet can be identified that should be resolved in any Information Superhighway that will carry advanced multimedia information. Bandwidth limitations must be overcome; the network must control the flow of traffic to avoid service interruption due to congestion, particularly for continuous services such as video; privacy must be assured whenever necessary to the application; security must protect the network from fraud and abuse; and the network operator(s) must be accountable to users if the network is to be depended upon. Finally, consideration must be given to the availability of guaranteed delivery of (multimedia) messages and to the possible need for a defined minimum functionality to all users, in order to assure that all will be able to send and receive certain kinds of message content.

The paradigm change from atoms to bits remains a demanding challenge.
Information Highway Applications *

D. De Roure
University of Southampton, Southampton, UK.

Abstract

High bandwidth network technologies are now available, both in the workplace and, increas-ingly, to the home. Here we focus on the applications that make use of this new technology. By taking a tour along a stretch of the highway—visiting the workplace, home and school—we identify some of the technical issues behind the application technology. We review the state of current practice, and finish by focussing on the Internet, and the World Wide Web in particular.

Keywords: Information Superhighway; Multimedia; Hypermedia; Internet.

1 Introduction

People talk about the information revolution, but the introduction of new technology is generally a process of evolution. Hence the first applications for the Information Superhighway are ‘net-worked’ applications which help us perform an established task, with some added value thanks to the better communication infrastructure. In the longer term the changes will be more profound.

In the next section we look at the impact of the superhighway on the workplace, the home and (because it is familiar to everyone) the educational institution. This highlights some of the technical issues which developers are working on now, and these are discussed in the subsequent section: multimedia, security, payment, navigation and resource discovery. For contemporary background discussion about the impact of the superhighway on our lives, see [1].

In the final section we look at the Internet, which for many is synonymous with the ‘Information Superhighway’, and in particular the Web (perhaps also synonymous with the Internet).

2 Some stops along the highway

2.1 At Work

Local area networks are fast and commonplace, but for a company split across multiple sites the wide area network connections have traditionally been much slower, and connections between companies far less common. Telephony and data are rarely integrated. With wide area network connections running at LAN speeds and above, and being available to all companies through service providers, improved practices are emerging:

- Email is used for information dissemination and asynchronous communication between individuals. Note that email uses a store-and-forward model and therefore does not require that a connection be established directly between sender and recipient, nor that the user is ‘online’ when the message is composed or read; store-and-forward is a robust model, but it does not make strong guarantees about delivery time.
- Videoconferencing enables people to meet without having to travel, and therefore facilitates more frequent meetings which can involve larger groups of people. It can also be used to ‘broadcast’ (or multicast) to groups of users, perhaps for information dissemination or for training purposes. Videoconferencing has been particularly successful in the

---

* Email dder@ecs.soton.ac.uk, http://www.soton.ac.uk/~dder/*
medical field, where local access to remote expertise is particularly valuable and the subject of discussion often involves visual information (e.g. X-ray images, scans).

- **Groupware** assists people working in groups, facilitating the collaborative tasks and perhaps supporting the management of the overall ‘process’, even across geographically remote sites. A shared workspace might involve document sharing, version management, and tools such as a shared whiteboard (this is classified as WYSIWIS—*What You See Is What I See*).

One downside of greater connectivity is a greater security problem, hence the increasing interest in **firewalls**: companies wish to make use of particular network services but protect their internal computers from the Internet. There is also increasing interest in encryption, to make information private and to provide authentication.

It is important not to forget the group aspects of applications; much of the discussion of the Information Superhighway tends to focus on individuals. The area of **computer supported cooperative work** is inter-disciplinary in nature and is as much about social science as it is computer engineering.

### 2.2 At Home

Terrestrial broadcast TV, satellite TV and cable TV all provide downstream bandwidth (i.e. to the user) but very little data flows back, and none between users. As upstream bandwidth becomes available, even if only at the bandwidth of voice grade phone lines, the situation becomes much more interesting: better interactive services are possible.

The current manifestation of the Superhighway at home is a fast modem (or possibly ISDN) and a general purpose computer. Meanwhile, set top boxes are becoming more sophisticated, and interactive TV is emerging [2]; even terrestrial TV will require set top boxes in the future, for digital services. The industry anticipates an increasing number of ‘information appliances’—domestic equipment connected to the network.

In recent years there has been a lot of ‘hype’ about the impact of the Information Superhighway at home. The applications commonly discussed include:

- **Video on demand.** From home you can select and watch a video programme, pausing it, forwarding and rewinding just as you would if using a VCR. The data is not stored locally in its entirety, but instead is transmitted from a video server at the cable TV *head end*. This requires communication from the consumer back to the video server, which is not widely available (because the amplifiers in the infrastructure are unidirectional). *Near video on demand* offers an interesting compromise which doesn’t need the back channel and therefore also works via satellite. Video programmes are transmitted simultaneously on different channels, staggered in time; consumers then have limited temporal control simply by changing channels, which only involves communication with the set top box.

- **Shopping.** Your TV or computer replaces the mail order catalogue, you have extensive search facilities, you can see videos of the products, and the transaction can be accomplished online. You can watch videos of you holiday location, or even see live views of it. There is clearly already a market for this, and (except the live videos!) it can be tested with CD-ROMs.

- **Information.** The superhighway gives access to a ‘universal encyclopaedia’, to electronic publications of magazines and books, to information services such as weather forecasts and online Yellow Pages, and to rapidly changing information (e.g. the stockmarket). How do you navigate the information space? —you need to find the information that meets your
requirements, presented in the way you want it. For example, your (online) newspaper might be customised to include just the news that you are interested in.

- **Telephony and videophony.** It has long been anticipated that telephone calls (‘audioconferencing’) will evolve into videoconferencing, and products have existed for many years but are not in widespread use. Quality video hardware is expensive, and this is now set to change with the home computer market acting as a driver for cheaper video technology. Fax is currently used to send images (note that even text is sent as an image) and should eventually evolve into multimedia mail, which includes sending video messages on a store-and-forward basis. Video mail can be more effective than text mail because much more information can be conveyed, and in some trials has proved more popular than live videoconferencing.

- **Teleworking (or telecommuting) enables people to work from home.** This is often achieved at present by direct connections to the company’s network, but is increasingly possible over the superhighway. Of course, not all tasks can be accomplished from home, and there are many problems with being away from the office. Teleworking can be particularly effective for people who offer their services to a variety of clients in different geographical locations. Modems over voice-grade phone lines are widely used; (narrowband) ISDN connections are prohibitively expensive in most countries.

### 2.3 At School

At present it is expensive for schools to join the highway; they are not in a position to enjoy the commercial benefits that would enable a business to connect. The importance of connectivity for schools is being acknowledged and there are now special deals and initiatives. There is also the issue, as at home, of protecting children from some of the material that may be accessible; British Telecom’s *CampusWorld* project uses the metaphor of a playground enclosed by a wall. Further and higher education establishments, on the other hand, have long been connected via academic networks (academics were early ‘citizens of the information society’) but only now are these networks becoming interconnected into the Internet.

The highway enables teaching material to be shared,¹ and it enables students to collaborate to enhance the learning process. Here we encounter the issues of *authoring*—how do teachers create new courses? This currently requires special skills, and tools to facilitate it are only emerging slowly.² It is not sufficient just to make educational information available, though this is an important first step; good course materials can also involve guided tours, question and answer sessions, simulations and assessment.

The video on demand technology, described above in the context of home entertainment, has an important role on campus in support of digital libraries. Users access a database machine which has an index to the available video material, and this in turn instructs a ‘media streamer’ to playback the video on the user’s workstation. The database machine controls access and can manage security, copyright and payment issues. The user cannot access the media streamer directly to select material, but may have a direct path for controlling playback.

---

¹ The *World Lecture Hall* can be found on [http://www.utexas.edu/world/lecture/](http://www.utexas.edu/world/lecture/)
² For educational technology, see [http://tecfa.unige.ch/info-edu-comp.html](http://tecfa.unige.ch/info-edu-comp.html)
3 Issues in application technology

3.1 Multimedia

A multimedia network, or a multimedia computer, is one which integrates different information types (usually represented digitally). In the multimedia context, the concept of a ‘document’ includes text, sound, graphics, image, animation or moving video; video and audio may be live rather than stored, and therefore need to be processed in real-time. The applications described above can be classified as multimedia applications. Some involve communication between people and are synchronous (e.g. videoconferencing) or asynchronous (e.g. multimedia mail). Teleworking and telebanking, on the other hand, involve communication between people and systems.

Standards for the storage and transmission of different media types have emerged, e.g. JPEG (for compression of images) and MPEG (for compression of video). JPEG exploits spatial redundancy. It defines a number of schemes for use in different circumstances, three of which are lossy; a lossy technique might provide compression of 24:1, while a lossless compression may be just 2:1. MPEG additionally exploits temporal redundancy. Blocks from a reference frame are matched against another frame, the best matches obtained, and it is the ‘prediction error’ and motion vectors for each block that are encoded. Forward and backward prediction is possible, as well as bidirectional prediction (interpolation). Standards are also required for the presentation of multimedia documents, where different information types are combined in a single presentation: this is the role of MHEG [4].

3.2 Security and Electronic money

In symmetric key cryptography (e.g. DES), the same key is used to encrypt and decrypt a message. Public key cryptography (e.g. RSA) uses asymmetric keys: a private and a public key are generated together (but cannot be derived from each other), and to encrypt then decrypt a message it is necessary to use one key then the other. The public key is published (but beware false keys!). By encrypting a message with the recipient’s public key, only the recipient can decrypt it (with their private key)—this provides privacy; by encrypting a message with the sender’s private key, anyone can decrypt it but having done so they know it was genuinely sent by the sender—this provides authentication (a digital signature).

The ‘ecash’ concept is based on digital coins, which can be stored on a disk and used in transactions in much the same way as real coins (including transfer between people). This approach overcomes a number of perceived problems with exchanging credit card numbers over the network; for example, as with coins, the payee need not know the identity of the person paying.

3.3 Navigating Information Space

With a huge volume of information available on the superhighway, the user needs tools to assist in discovering the appropriate information, and in searching the vast information space interactively or automatically.

Hypermedia has proven to be a powerful technique for navigation. From locations in one document, the user can follow links to related documents. These links might start from arbi-

---

3 For MPEG documents see ftp://ftp.crs4.it/mpeg/

4 For links to various publications, see http://www.digicash.com/
trary selections by the user, in which case the user is effectively querying the system for related information, or they could be associated with the document and more prescriptive about the interaction (as in a kiosk).

In open hypermedia, the link information is stored external to documents, which can remain in their native format. This has a number of significant advantages for many applications. It eases maintenance of links, it facilitates authoring (anyone can create links, in link databases which are existing or new, private or shared), and enables the sets of links available at any time to be customised to the user’s requirements. Publishers can then use the same set of ‘information assets’ and repurpose them to meet the requirements of different publications.

4 The Internet

4.1 Background

The history of the Internet goes back to the early 60s (see [3] for a thorough account of the history and the design of Internet protocols). During the 1980s a number of resource discovery tools emerged (gopher, wais, archie) to assist in finding information on the increasingly vast network; meanwhile users were typically performing file transfers to obtain or publish information. The arrival of the Web and its browsers made these tasks much simpler for the user; it is this simplicity of use combined with the availability of the browsers, that is probably responsible for the widespread popularity of the Web. Existing resource discovery tools have been integrated with the Web, and new techniques are being developed.

4.2 Internet protocol over ethernet

Here we introduce some of the concepts underlying Internet networking, by looking at Internet protocols over ethernet.

A packet travelling from host to host on ethernet carries the source and destination ethernet addresses. An IP (Internet Protocol) packet also carries the source and destination IP addresses (each 4 bytes). The ethernet addresses correspond to the current ‘hop’ only, while the IP addresses correspond to the extreme endpoints of the communication. The movement of the packets will be controlled by active equipment in the network infrastructure: bridges filter traffic according to the ethernet address, routers work at the level of the IP addresses, and gateways at higher (application) levels.

When an IP packet is transmitted, the host checks to see if the destination IP address is on the same physical network, and if so it uses the corresponding destination ethernet address. However, if the destination IP address is not on the same network, a suitable router (often called a gateway) is identified from routing tables; it is the ethernet address of this machine that is used as the destination ethernet address, while the destination Internet address remains unchanged. Two IP addresses are identified as being on the same network if they have identical network portions (obtained by applying the network mask to each, e.g. with a mask of 255.255.255.0, hosts 152.1.2.3 and 152.1.2.4 are found to be on the same physical network).

To determine the ethernet address of a host from its IP address dynamically, the ARP (Address Resolution Protocol) is used. Usually only the host itself would respond to a broadcast request for its ethernet address (if two machines respond with different ethernet addresses then there are duplicate IP addresses in use). There are some circumstances where one machine may respond on behalf of another; for example, a router might respond with its own ethernet address for all the IP addresses on the other side of it (this is called proxy ARP, and generally routing is
preferable where possible).

Above the IP layer, there are connectionless protocols (using UDP—User Datagram Protocol) and connection-oriented protocols (using TCP—Transmission Control Protocol). For example, the network file system (NFS) uses Remote Procedure Calls (RPC), which use External Data Representation (XDR) and UDP; i.e. it is not (traditionally) a connection-oriented service, and this enables it to be stateless but also makes it less secure.

We use a number of connection-oriented Internet protocols on an everyday basis. The services listed below are given with their port numbers, which are well-known numbers used by client software to access that particular service on a given host.

- SMTP (Simple Mail Transfer Protocol, port 25). This is the protocol used to send email messages from host to host, and may be used when you send messages using your mail program (user agent).
  
  MAIL From: dder
  RCPT To: someone
  DATA
  Greetings from the CERN Summer School of Computing

- POP (Post Office Protocol, port 109) is used by some mail programs to read spooled email.

- FTP (File Transfer Protocol, port 21). This is the protocol used by the `ftp` command for interactive file transfer.

  USER dder
  PASS secret
  CWD pub
  RETR file.txt
  QUIT

- TFTP (Trivial File Transfer Protocol) is a lightweight file transfer protocol based on UDP instead of TCP; it implements its own acknowledgment protocol.

- HTTP (HyperText Transfer Protocol, port 80). Used in communication between World Wide Web clients and servers. The familiar `methods` are GET (for obtaining documents and sending simple queries) and POST (for sending data such as a form).

Two protocols for IP over serial lines are being used increasingly, for dial-up connections to the Internet: SLIP (Serial Line Internet Protocol) and PPP (Point-to-Point Protocol). SLIP is the simpler and is quite widely supported in public domain products; there is also a version which uses compression. PPP supports multiple protocols, handles address negotiation (useful for mobile hosts) and has an authorisation mechanism; it is currently more often available in commercial products. Multilink PPP is also appearing and enables a single logical PPP channel to be established over multiple network channels, such as in aggregation of 64kbps ISDN channels.

### 4.3 World Wide Web

World Wide Web is an example of a distributed hypermedia system, running on the Internet. It is very simple: the user uses a browser, which is a tool than can display a number of standard document formats, to access a server, which provides file transfer. The client and server communicate using HTTP. Hence the user can download a file and display it, by instructing their browser to get the URL which identifies the document. One of the formats supported by the browser is HTML, which is a mark up language that includes hard coded hypermedia links from
locations in the document to other documents. The server maps filename extensions to particular MIME (Multipurpose Internet Mail Extensions) encodings, and the browser in turn maps these to corresponding viewer applications.

To facilitate searching and interaction through forms, it is possible for the server to initiate a process to handle an incoming query or form, using the Common Gateway Interface. Similarly, the browser can create other processes (perhaps to handle certain document formats) and has the capability to be integrated with other applications. Server-side state does not persist from call to call, so where required the state must be encapsulated in the document sent back to the client. Netscape implements the notion of a cookie sent by the server and retained by the client, to be inserted automatically in future requests in a given range of URLs.

The Web model is strict client-server. It is possible to introduce processes called proxies which the clients treat as servers; meanwhile, the proxy acts as a client to other servers. Caching proxies enable a group of clients to enjoy local access to documents of common interest once the document has been obtained for the first time. The relevance of proxies here is that they promote scalability: when moving an application from a LAN of 100s of machines to what seems to be a global LAN of millions, scalability is a key issue.

As it stands, the Web lacks many of the features needed to tackle the applications on the Information Superhighway. Security is improving (e.g. Netscape SSL) and the restrictions of the strict client-server model are being tackled so that it supports more asynchronous interaction (e.g. client-pull and server-push). It does not currently support real time video, as in videoconferencing, but can be used to integrate such applications (there is support for video on Internet, through the Multicast Backbone or MBONE). As the browsers become more programmable (with languages such as Java) there will be more control over presentation and interaction, with more processing client-side. VRML (Virtual Reality Modelling Language), which in some ways is like HTML but describes 3D models, is also becoming available.

The Web is currently used as a closed hypermedia system, because the links are stored in HTML documents. However, it has an open architecture and it is possible to engineer open hypermedia solutions. For example, links can be stored external to the documents, and then either (a) precompiled them into HTML documents before the user accesses them, or (b) compiled into the document on-the-fly, when it is requested by the user. The latter has the disadvantage that server-side processing is required at the time of the document access, so the document cannot be held in caching proxies. With some additional client support, authoring of link databases by simple point-and-click operations is possible, and in combination with the facility to search existing link databases, a powerful authoring environment is achieved.5

4.4 The future

Many people refer to the Internet and the Information Superhighway interchangeably, while for some the Information Superhighway is the world of CATV and video on demand (the so-called ‘500 channels’); also, we should not forget the developments in mobile phones and new wireless products. The Internet is highly interactive, highly programmable, open, general purpose and somewhat anarchic; at the moment, the Internet infrastructure uses the public networks but the Internet is not operated by them per se—they do not dictate the information content. As such it seems to form a stark contrast with the cable world, where the company decides on what information to make available and then sells it to the ‘consumers’ along with other services for ‘information appliances’. These are issues in the ongoing debate about the future model of the

5See http://www.cosm.ecs.soton.ac.uk/dls/
Information Superhighway.

Acknowledgements

Some of the material in these lectures is based on the *Networking in the Nineties* module of the *Modular MSc in Information Engineering*, a course delivered using the Internet. 6

References


6http://modmsc.ecs.soton.ac.uk/
Brief History of Computer Architecture Evolution and Future Trends

R. D. Groves
IBM, Austin, Texas

Abstract

Computer system architecture has been, and always will be, significantly influenced by the underlying trends and capabilities of hardware and software technologies. The transition from electromechanical relays to vacuum tubes to transistors to integrated circuits has driven fundamentally different trade-offs in the architecture of computer systems. Additional advances in software, which includes the transition of the predominant approach to programming from machine language to assembly language to high-level procedural language to object-oriented language, have also resulted in new capabilities and design points. The impact of these technologies on computer system architectures past, present, and future will be explored and projected.

General

The key hardware technologies that affect computer architectures are those that determine the density and speed of digital switches and the density and access time of digital storage. Figure 1 shows the trend in increasing capacity and performance for digital switching technologies. The important trend to notice is that capacity improves at a more rapid rate than transistor speed. Figures 2 and 3 show the trends in capacity and access time for two important digital storage technologies: dynamic random access memories and magnetic disk storage. Here again, the important trend reflected is that capacity is improving at a faster rate than time taken to access the data stored. These basic trends have been true throughout the history of computer systems and are projected to continue through the foreseeable future in the absence of fundamentally new technological approaches.

Despite these underlying trends, the performance of computer systems has increased at a rate which exceeds the rate of improvement in transistor speed and digital storage access speeds. To achieve this, computer systems have been required to take approaches that improve performance by exploiting both technological density and speed. Since these basic trends are projected to continue, future system designs can be expected to leverage both density and speed to achieve additional system performance.
Figure 1: Using their 1983 capabilities as a baseline, transistor speed and the number of available CMOS circuit elements per die is plotted over time. Source: IBM and [1].

Figure 2: Using their 1980 capabilities as a baseline, the row access performance of DRAM and the DRAM capacity is plotted over time. Source: IBM and [2].
In the earliest computer systems, both density and speed were quite modest; and software technologies were very immature. These facts drove computer architectures, which used very simple instructions and register sets combined with machine level programming, to provide the maximum in efficient exploitation of the limited number of switches and storage capacity. As hardware became denser and faster, the number of registers and the complexity of the instruction sets were increased. Advancing software technology allowed programming to move to the assembly language level and, eventually, to high-level procedural languages.

In the late 1950's and early 1960's, the gap was wide enough between memory technologies and processor speeds that functions implemented as instructions would execute as much as ten times faster than those programmed as function or subroutine calls. Compiler technology was still in its infancy and efficient compilation still eluded the general programming environment (though many early FORTRAN compilers were demonstrating quite astounding results for this time period). These technological constraints drove computer architects to provide ever more complex instruction sets. More complicated instructions helped overcome the memory-processor performance gap by doing more "work" for each instruction fetched. Many architects were also convinced that the job of the compiler could be simplified if more complex instruction sets could be used to close the "semantic gap" between the high-level languages and the machine-level instruction set.

As this trend toward more complex instructions continued into the 1970's, the underlying technologies were changing to the point where a fundamentally new set of design trade-offs was
possible for optimum system performance. Complex instructions were actually implemented using a high-speed, read-only control memory that actually interpreted the complex instructions and executed a "microprogram" contained in this high-speed storage. As instruction sets became more complex, this control store became larger and more complex as well. By the 1970's, the complexity reached a point where the correctness of this microcode could not be adequately assured. Thus, most computer systems began using a writeable control store so that corrections could be made in the field as errors were discovered and corrected. At about the same time, the concept of using high-speed, writeable memories, called caches, to contain the most recently used instructions and data proved quite effective in reducing the average latency (access time) to the memory subsystem because of the ability to exploit the temporal- and spatial-locality of most memory accesses.

Compiler technologies were advancing as well. Compiler optimization technologies were proving to be most effective when certain simplifying assumptions were used, including the use of very simple operations in the intermediate languages (IL) for performing the optimizations against. By using very simple IL operations, all operations were exposed to the optimization algorithms resulting in the dramatic reduction in the amount of code generated. These optimizations often resulted in code as good or better than that generated by average assembly language programmers. Another key set of technologies for compilers involved algorithms which allowed efficient utilization of the storage hierarchy. The use of registers has always been problematic for compilers. The register allocation problem was dramatically improved by the introduction of register coloring approaches [4]. The efficient exploitation of the cache and memory hierarchy became possible with the advent of cache-blocking techniques and loop transformations which enabled them.

From an operating system perspective, the trend towards portable operating systems, of which UNIX is the most obvious example, was creating a system design point in which most of the software in the system would be written in a portable, high-level language with relatively little code specifically written for the specific hardware system and almost no code written in assembly language. Combine all of these changes with the dramatically increasing density of the underlying hardware technologies and the fundamentals all pointed to a major change in computer system architectures. Several seminal research projects were undertaken in the mid to late 1970's based on these observed technological changes, including the IBM 801 [5], Berkeley RISC [6, 7], and Stanford MIPS [8] projects.

These projects challenged all of the existing fundamental concepts and approaches being used relative to the trade-offs between hardware and software. As often happens in major changes, these projects took the extreme opposite view of assuming that everything should be done in software unless proven otherwise. Thus, these projects explored all the boundaries of these design trade-offs. Fortunately, many of these proposed approaches never left the laboratory. For example, at one point the 801 project was attempting to eliminate the need for storage protection hardware by relying on a trusted operating system and compiler using a language which did not allow unbounded pointers. Theoretically this was a very nice concept, but not practical in the open systems software environment. Another example that all three projects explored was whether all instructions could be restricted to a single cycle operation. This implied that important functions like integer multiply and divide as well as almost all floating-point operations would have to be programmed from simpler,
one-cycle primitives. By the time these architectures were commercialized, these more "complex" operations were included. With the increased density of circuits available, the number of cycles required for these operations had reduced significantly (with many of today's RISC CPUs able to perform independent floating-point operations every cycle); and the overhead associated with making subroutine calls to special primitive instructions to perform these functions would have significantly impacted the overall performance for these operations.

The resulting commercialized RISC architectures included IBM POWER (based on the 801), Sun SPARC (based on Berkeley RISC), Silicon Graphics MIPS (based on Stanford MIPS), Hewlett Packard PA-RISC, and Digital Equipment Alpha. All of these architectures share some common characteristics. All have fixed-length (32-bit) instructions with relatively few formats making instruction decode extremely easy (sacrificing information density for ease of decode). Each has 32 general purpose registers as well as at least 16 floating-point registers. All have at least 32-bit addressing with extensions to 64-bit addressing (Alpha being the only one which supports only 64-bit addressing). Data references are restricted to explicit load and store instructions which transfer data from/to memory to/from an internal register. With the exceptions of operations such as integer multiply/divide and floating-point operations, the rest of their instruction sets are optimized around simple, single-cycle instructions. Separate instruction caches are used to supply the necessary instruction bandwidth and to provide a logical replacement for the high-speed microcode control store that previous machines used to contain microprograms. The advantage of an instruction cache is that the code for the most frequently used functions automatically migrates to the instruction cache as opposed to a microcode control store where the functions contained there are predetermined by the instruction set architecture.

These features all reflect the fundamental philosophy of creating a simple, straightforward instruction set that can be implemented efficiently in hardware and lends itself to efficient compiler optimization. Initial implementations of these architectures were able to achieve competitive instruction path lengths and to execute nearly one instruction every cycle for a wide variety of interesting workloads of which the SPEC benchmarks are representative. Workloads that are more stressful on the memory hierarchy, similar to that exhibited by the NAS parallel and TPC-C benchmarks, are dominated by cache and memory effects rather than the raw CPU capabilities and require careful attention to the design of the cache and memory subsystem. Nevertheless, the initial systems provided very attractive performance and price/performance, particularly in engineering/scientific applications, and established a significant foothold in the marketplace.

Some, less than optimal, trade-offs did survive the transition from the laboratory to commercialization. Most of these systems made some simplifying assumptions relative to cache management and coherency with I/O. The IBM POWER architecture took this to the extreme of requiring all cache-to-I/O coherency to be managed in software. While this has been made to work efficiently for the common case of paging I/O, the overhead for "raw" I/O is significant and complicated. Thus, cache-to-I/O coherency is now supported in hardware for PowerPC. The Sun SPARC architecture supports a feature called register windows which keeps the context of several layers of the subroutine-call hierarchy in registers. While this approach does reduce the overhead associated with subroutine calls and returns, it does restrict optimization across subroutines and
increases the overhead of context switching. The overall benefits are small and do not justify the additional hardware circuits and complexity.

After the initial round of implementations, computer architects realized that most future performance gains would have to come from the exploitation of parallelism. Since CPUs were already executing close to one instruction per cycle, without parallelism, almost all future performance gains would be limited to that obtained from faster circuit switching speeds. Exploitation of parallelism leverages the increasing circuit densities to achieve system performance growth which exceeds that of the underlying switching speed improvements. The forms of parallelism that have been explored include: instruction-level parallelism (executing more than one instruction per cycle), task/process-level parallelism (simultaneous execution of more than one task or process), and algorithmic parallelism (dividing up a single problem so that it can be executed simultaneously on more than one processor). Each of these approaches and their current status and future potential will be discussed.

Instruction-level Parallelism

After the initial RISC implementations, the focus of computer architects began to move from the current state of the art of nearly one instruction per cycle to multiple instructions per cycle (or superscalar implementations). This was driven by the fact that doing more operations in one cycle would allow the exploitation of additional circuit densities to increase system performance above that achievable by mere circuit switching speed improvements. Many of the architectural decisions made earlier mapped very well to superscalar implementations. The regularity of the instruction set and formats, the large number of general purpose registers, and the relatively few complex operations all greatly simplified the task of resolving interdependencies and resource conflicts necessary to dispatch and execute multiple instructions in one cycle.

However, some of the early RISC architecture decisions proved problematic in the move to superscalar implementations. The most obvious was the inclusion of the "delayed branch" in these architectures. Since a taken branch introduced a "bubble" of one cycle in the pipeline of these early processors, the delayed branch allowed the compiler to insert a useful instruction after the branch that could be executed during this otherwise idle cycle. Compilers were typically quite successful in finding instructions that could be placed in this delay slot; and this feature was very important for achieving the nearly one instruction per cycle goal of these systems. However, this feature did add complexity to the design since the delayed branch and its subsequent instruction often had to be treated as an atomic pair of instructions creating extra complexity especially under error conditions. On the other hand, superscalar implementations usually process branch instructions separately looking ahead into the instruction stream so that taken and untaken branches are essentially eliminated from the pipeline. Thus, not only is the delayed branch not needed, but it also adds complexity for no performance gain. This is why later RISC architectures like IBM POWER and DEC Alpha removed the delayed branch from their architectures.
Another area of concern with superscalar implementations is that performance tends to be ultimately limited by the ability to deal with the changes in control flow. Average instruction streams contain a taken branch every 5 instructions; so, to achieve any reasonable levels of parallel instruction execution requires significant amount of resources either resolving or predicting the outcome of branch instructions. Control flow is typically predicated on the testing of condition codes. Architectures which provide the ability for multiple, independent condition codes allow the compiler the opportunity to set and resolve them as early as possible in the code, greatly improving the accuracy of the branch resolution and prediction capabilities and simplifying what must be implemented for the same level of performance. Most RISC architectures were not designed with this in mind and have mixed capabilities in this area. Most have a single condition code, or worse, the MIPS and Alpha architectures combine the compare and branch into a single instruction. These architectures do provide the ability to place conditions into a general purpose register which is inefficient (only one bit of information in a 64-bit register), uses precious register locations that could be used for other computations, and adds complexity in branching logic which must then share these registers with the integer unit. The IBM POWER architecture is unique in having architected multiple, independent condition codes for use by the compiler as well as complete control of the setting of these condition codes in the instruction set.

Since the introduction of the first superscalar RISC, the IBM POWER microprocessors in the RISC System/6000 in 1990, increasingly aggressive superscalar implementations have appeared from all key vendors and have delivered even more impressive performance and price/performance particularly in engineering/scientific applications where the ability to execute fixed-point and floating-point instructions in parallel provides the ability to execute 3 to 6 instructions per cycle sustained with today's implementations. For a brief period, a few "superpipelined" processors were introduced in the market (most notably the MIPS R4000 and, to some degree, the original DEC Alpha which was both superscalar and superpipelined). The approach here was to exploit parallelism by making deeper pipelines instead of more parallel functional units. Since this approach does not exploit the extra density of circuits and puts extra stress on the circuit switching speeds, it has rapidly fallen out of favor.

Despite the success of superscalar RISC processors, the ability to efficiently exploit more instruction-level parallelism with purely superscalar approaches is reaching diminishing returns. This has given ascendancy to very long instruction word (VLIW) techniques [9]. The fundamental limit for superscalar designs is instruction dispatch. To exploit additional instruction-level parallelism requires an ever more complex instruction dispatch unit that can look past multiple branches and determine what instructions can be executed, or speculatively executed, on a given cycle based only on information available at run time. This is a geometrically complex problem as the number of instructions to be dispatched simultaneously increases. VLIW approaches move the determination of which instruction to execute, or speculatively execute, in a given cycle to the compiler based on compile-time information which is often more complete. The hardware implements a large number of parallel functional units, but only executes those operations in parallel that the compiler places into a single very-long-instruction word.
VLIW architectures which leverage large numbers of parallel functional units require more than the 32 registers typical of today's RISC architectures. They also fundamentally require the ability to resolve multiple branches in a single cycle implying the need for multiple condition codes.

Major advances in the state of the art of VLIW hardware and software have been made in recent years, but many issues remain unresolved. The biggest issues are compatibility with existing RISC architectures and the ability to support a range of VLIW implementations with the same compiled binary code. Now that RISC computers have established themselves in the industry, market acceptance of a new VLIW machine would be greatly enhanced by being able to support all existing applications without requiring a recompile. (In fact, this may be necessary for them to be commercially successful.) Also, a VLIW architecture must be able to support differing levels of parallelism in the underlying implementations to allow multiple price/performance points and to exploit increasing density in the future. This remains a field of very active research with all major players working to determine how to best exploit VLIW technology in their future systems.

Independent of VLIW or superscalar processors, the problem of "finite cache effects" is becoming an increasingly important aspect of system architecture and design. VLIW and superscalar processors can achieve very impressive levels of instruction-level parallelism as long as their data and instructions are contained within the high-speed caches connected to the processors. However, with the levels of parallelism being achieved, the number of cache requests per cycle is increasing; and the latency for resolving a cache miss is also increasing in relative terms. As figure 4 shows, the CPU performance is increasing at a faster rate than the access time of main memory creating an increasing performance gap. All of this is driving system designs which include multiple levels of caching; multiple, simultaneous outstanding cache misses; and cache line prefetching approaches. All of these focus on ways to decrease the average latency of cache misses and reflects incorporating parallelism and speculation into the cache subsystem to match the parallelism and speculation that is already in the processors. Algorithms and hardware implementations that are able to better tolerate latency will become increasingly important in the future.

**Task/Process Parallelism**

An obvious source of parallelism is the multiple, independent tasks and processes which run simultaneously on most computer systems. This level of parallelism has been exploited naturally by Symmetric Multiprocessor (SMP) systems by running independent tasks or processes on separate processors. This is a trend that will continue. Recently, interest has been revived in exploiting task/process parallelism at the individual processor level. Early CDC I/O processors were actually a single processor that was time-shared between multiple, simultaneous contexts on a
cycle by cycle basis. In a similar fashion, much research is being done on sharing multiple contexts within a single processor with a very lightweight mechanism for switching contexts (on the order of a single cycle). Context switches would be initiated when a long running operation is encountered such as a cache miss. This would allow useful processing to be performed in other contexts during the latency of the cache miss, thus improving the efficiency of CPU utilization. For these approaches to be effective, the cache subsystems must be capable of supporting multiple, outstanding cache misses.

**Algorithmic Parallelism**

Task/process parallelism is able to improve the throughput of a computer system by allowing more tasks/processes to be completed per unit time. Algorithmic parallelism attempts to improve the turn-around time for a single problem or task, and may actually negatively impact throughput. Algorithmic parallelism involves rethinking the fundamental algorithms being used to solve a problem on a computer to come up with a new approach which will be able to be efficiently divided among several independent processors to work on the problem simultaneously. Some problems
require very little rework to accomplish this, though most require significant efforts to create an efficient and scalable approach. Almost all problems require significant redesign of algorithms to exploit high levels of multi-processor parallelism.

The fundamental limit of parallelism and scalability is the level of sharing required between the processors. Sharing can exhibit itself at the instruction, data, or I/O level. In actual implementations, sharing can also occur because buses, cache lines, or other implementation artifacts are shared between the processors, but are not sharing constraints imposed by the algorithms. Computer system designs are exploiting three approaches to reduce the amount of sharing at the hardware level, while still providing an efficient environment for the current state of the art in applications and operating systems. These three types of multiprocessors have been called: uniform memory access (UMA), non-uniform memory access (NUMA), and no remote memory access (NORMA).

**Uniform Memory Access**

This represents the sharing model of most commercially successful symmetric multiprocessors (SMPs). They are characterized by all processors having equal access to all memory and I/O. They typically run a single copy of the operating system that has been altered to exploit parallelism and to give the impression of a uniprocessor to the users and all uniprocessor applications. Applications that wish to exploit parallelism are provided with a set of programming interfaces, services, and tools to be able to execute on multiple processors simultaneously.

While UMA MP technology is quite mature, it does have some drawbacks for the exploitation of algorithmic parallelism. The fact that memory, I/O, and the operating system are all shared by all the processors creates a significant amount of implicit sharing which fundamentally limits the scalability of these designs. This problem only becomes worse as the speed of each individual processor increases. This will drive SMP systems to move from today’s dominant design point of shared-bus systems to systems in which the connection among the CPUs and memory and I/O consists of various switched media to reduce the amount of implicit sharing. Further hardware enhancements will be added which speed up the cases where the algorithms themselves require sharing. Even with all these valiant efforts, the limit of scalability of UMA systems will remain in the low double digits.

In addition to performance scaling problems, UMA MP systems also suffer from system reliability problems. Because memory, I/O, and the operating system are all shared between all the processors, any failure in any component is highly likely to bring down the whole system. Thus, reducing the level of sharing can also improve the availability of a system, as well.
Non-Uniform Memory Access

NUMA machines share a common address space between all processors and their memory and I/O; however, the access time from any given processor to some of the memory and I/O in the system is noticeably longer. That memory and I/O with the shortest access time is usually referred to as "local" to that processor. Memory and I/O which is farther away is either referred to as "global" or "remote". Usually the bandwidth is higher to local resources than it is to remote resources.

By relaxing the requirement for uniform access, the hardware system designers are more easily able to construct systems which scale to much larger numbers of processors. Uniform access places significant physical constraints on the design since the speed of light ultimately determines access times. Thus, uniform access designs eventually reach a point where processors and memory cannot be packed more densely without increasing access times.

NUMA designs will function well provided most accesses can be made to local resources. In fact, the average percentage of remote accesses is the ultimate determinant of scalability in NUMA systems. Thus, the scalability of NUMA systems is highly dependent on the ability of the application and the operating system to reduce sharing. Some of this is accomplished by replication. For read-only data (or instructions) that are shared, a replicated copy can be placed locally everywhere it is being used and all references can be made local. Data (or instructions) that are not shared only exist locally by default. The challenge comes in reducing the amount of sharing of data that is updated and read by more than one processor.

Further complicating the situation is the fact that since all addressing is common, but data is allocated in discrete sizes (such as pages or cache lines), false sharing can occur by having data that is not being shared having been allocated to addresses too close together. Also, the realities of trying to balance processing loads among the multiple processors requires that processes or tasks must sometimes be migrated to other processors. This migration requires that the associated local data be moved also. Since the operating system and hardware are attempting to manage this at run time with only run-time information, this is a difficult task in the general case.

The design of the operating system for NUMA systems has two approaches. The first is to modify an operating system designed for UMA machines to deal with replication of memory and remote/local optimizations. This will ultimately entail replicating many operating system services to more distributed approaches such as having local task dispatch queues coordinated by an overall global task dispatch algorithm. The advantage of this approach is that a single system image is preserved that helps applications in the transition from UMA machines. This is the approach that has been taken by most NUMA machines in the marketplace.

A second approach is to run separate copies of the operating system locally (i.e. replication of the operating system); and then provide global services that each operating system copy uses to manage the resources of the whole system. This approach has the advantage of starting from a design point where no sharing exists and adding sharing and cooperation as needed through the global services. The downside is that since all services are not global, the migration from UMA machines is
complicated. Furthermore, modifying the operating system to be able to cooperate on controlling global resources is usually non-trivial.

The common thread here is that NUMA machines provide scalability by reducing sharing through the use of replication. Since replication explicitly uses more memory, this is another example of exploiting density improvements to improve performance.

**No Remote Memory Access**

NORMA machines do not share memory, I/O, or operating system copies. The individual nodes in a NORMA machine cooperate with other nodes through the sending of messages. High-performance NORMA machines focus on improving the bandwidth and latencies of sending messages between nodes particularly at the application level. Much like the second approach mentioned for NUMA operating system design, global services and resource allocation are provided by a layer of software which coordinates the activities of the multiple copies of the operating system. Applications must be structured and written to exploit message passing as opposed to shared-memory communication constructs and to use only those services that are globally enabled by the coordinating layer.

The NORMA approach is clearly the most scalable from a hardware perspective and has demonstrated this capability for applications which have been structured to have limited sharing, as well. However, most applications have yet to make this transition. To assist in this complex process and to improve performance, NORMA machines are beginning to provide "remote memory copy" operations which allows an application to copy to/from another application's address space in another node on the system in a very lightweight fashion. This capability will further close the semantic gap between shared-memory programming and message-passing programming and allow compilers and application writers to be able to more easily design and move applications to NORMA machines. Also, the move to object-oriented application design lends itself well to NORMA architectures since method invocations are essentially messages.

In fact, if one looks at what is required to implement an efficient NUMA machine and a NORMA machine that supports remote memory copy, much of the hardware required is quite similar. Both machines consist of nodes with local memory and I/O interconnected through some kind of fabric to other nodes. In a NUMA machine, data is transferred from one node to another based on a cache miss to a remote node. In a NORMA machine, data is transferred between nodes at the request of an application (with a PUT or GET command). The actual process of doing the data transfer is very similar in hardware. The key difference is whether the transfer was initiated by hardware (a cache miss in a NUMA machine) or software (a PUT/GET in a NORMA machine). Because of these similarities, some of the current research projects are attempting to design machines that are capable of supporting both simultaneously.

Again, NORMA machines achieve performance by exploiting replication, even more so than NUMA machines. This, of course, leverages increasing density for more performance.
Summary

Computer architectures have evolved to optimally exploit the underlying hardware and software technologies to achieve increasing levels of performance. Computer performance has increased faster than the underlying increase in performance of the digital switches from which they are designed by exploiting the increase in density of digital switches and storage. This is accomplished by the use of replication and speculative execution within systems through the exploitation of parallelism. Three levels of parallelism have been exploited: instruction-level parallelism, task/process-level parallelism, and algorithmic parallelism. These three levels of parallelism are not mutually exclusive and will likely all be used in concert to improve performance in future systems.

The limit of replication and speculation is to compute all possible outcomes simultaneously and to select the right final answer while discarding all of the other computations. This is analogous to the approach DNA uses to improve life on our planet. All living entities are a parallel computation of which only the "best" answers survive to the next round of computation. These are the concepts driving research in the area of using "biological" or "genetic" algorithms for application design as well as the research into using DNA to do computations. In the limit, molecular-level computing provides huge increases in density over current systems, but much more limited improvements in raw performance. Thus, we can expect continued advancements in those approaches and architectures that exploit density and parallelism over raw performance.

References

High-Speed Switching

Ton Engbersen
IBM Research Division, Zurich Research Laboratory, Rüschlikon, Switzerland.

Abstract

This lecture introduces switching by viewing the development of switching technology from a historical perspective. Circuit switching, packet switching, and modern combinations of these two basic technologies are presented. Extensive discussion is devoted to the performance aspects of switches, and how to deploy switches in a network. The lecture also includes an in-depth handling of ATM switch technology in part 2.

1 A historical perspective.

The first applications of switching were in the field of telephony. The increasing use of telephones prompted the use of ‘exchange centers’, where connections with other telephones were made by human operators. Actually, the word Operator was first used in this context, and at least in the US, is still used today. The function of these manually operated switches is shown logically in Figure 1 ”Basic Circuit Switch Principle’. This figure also shows one of the main functions of these early switches: to make connections between two subscriber lines, or, from another perspective, to establish a copper circuit between two subscribers. In fact, for the duration of their conversation, the two partners in a phone connection ‘own’ the piece of copper connecting them. No one else can make use of that piece of copper. This method of switching later became known as Circuit Switching. We call the center of these switches the ‘crossbar’. This type of switching became established early in the history of the phone network and, for a long time it remained virtually unchanged: the Operator was a well-known part of the phone network who made the connections between subscribers. Increasing traffic prompted the development of more automated equipment, and this resulted in the circuit switch. As switching needs grew, the demand for a formal background became evident and, in a landmark paper published in 1953, Clos [2] described ways to extend switches to larger entities using identical switch-building blocks.
2 Introduction of data communications.

During the 1960’s, computer use was often organized in the form of large computer centers, and access to these computers via remote terminals, using a new method called time-sharing, became fashionable. Via telephone connections, remote terminals were connected to these computing centers, and modems translated the digital information into analog signals for transmission over the phone system. Although this system appeared to function very well, a more detailed investigation of the actual mechanism reveals serious inefficiencies. Figure 2 "Analysis of Packets over a Circuit Switched Network’ shows what actually flows over each connection in time. The low interaction rate of the terminal user(s) results in a poor utilization

![Figure 2: Analysis of Packets over a Circuit Switched Network](image)

of the composite of connections, and the circuit characteristic of the connections do not allow users to be multiplexed effectively over one line or circuit. A new technology was then invented: Packet Switching. In packet switching, each piece of information is prefixed by a header, which supplies information to the network concerning how to handle (i.e. route) the subsequent data. This mode allows the efficient handling of typical bursty computer traffic over one physical connection. However, it also creates new problems: can the use of the physical medium be optimized by statistical multiplexing? If so, what happens when the traffic temporarily exceeds the physical capacity of the circuit? A special role in this mode is played by the switching technology: in circuit switching, the switch had to establish a physical connection during the call-setup time, and maintain the connection until it was no longer needed; in packet switching, the switch has to inspect every incoming packet header, interpret the contents, and switch the remaining data, plus a potentially changed header, to the proper switch output, all in the presence of statistical multiplexing.

3 Packet switching architectures.

Two tasks must be accomplished by a packet switch: the header must be interpreted/modified and the packets must be routed to the output(s). The statistical nature of the traffic requires that routing and header processing must be accomplished at the aggregate rate, i.e. the sum of the line rates of all incoming communication lines. As long as this aggregate rate is low, a simple processor/bus/memory system is sufficient: packets move from input line-adapters into main memory, are accessed by the processor, inspected, and dispatched to output line-adapters. Note that the packets pass the system bus twice, which limits the packet throughput. Direct input-
adapter to output-adapter(s) communication over the system bus removes this bottleneck. The fact that all adapters on the system bus require interfaces that operate at the aggregate throughput (which easily reaches several Gigabit/sec) means that these systems will not migrate easily to higher speeds in the future: any extension of the system to accommodate more adapters requires the redesign and replacement of all existing adapters.

By making use of space-division mechanisms, systems can be built that support a much higher aggregate throughput. These systems can also be extended to support higher aggregate speeds without adapter redesign [3]. With regard to throughput, this case also requires that the aggregate speed be supported under all conditions. Note that a simple crossbar switch does not fulfill this requirement. Only when the arriving packets do not compete for identical outputs can a crossbar sustain the aggregate. By adding appropriate input queuing, it is possible to alleviate this problem to some extent, depending on the expected traffic characteristics. The proper solution is to sort and queue the packets at every input according to destination, and to develop a controller that dispatches the maximum amount of traffic at all times to the crossbar. This, however, is a design that cannot easily be extended to larger switches: the controller complexity grows at a rate of $N*M$. ($N=$number of inputs, $M=$number of outputs).

Another method is to provide queues at the outputs that can be written to at the aggregate input rate. Performance evaluation shows that already a small amount of memory used as a queue per output results in a significant increase of sustainable throughput, i.e. an output queue memory is highly effective. This creates a completely new opportunity: to integrate a small amount of queue memory together with appropriate control on a single chip. In fact, single-chip packet switches become feasible when based on this output-buffered architecture.

One final observation should be made, however. In the output-buffered architecture described here, there are $M$ queues, each supporting the aggregate input rate. Hence the $M$ queues would support $M$ times the aggregate input rate. This is a factor of $M$ too much! Realizing that the individual queues start to queue packets when more than one input is sending a packet to the output to which the queue belongs, we can propose a solution. By sharing all the queue memory among all outputs, a heavily loaded output can 'borrow' queue space from the necessarily less heavily loaded output(s). Besides allowing an even more efficient use of the memory available, a shared memory needs only to support exactly the required aggregate throughput. Important operational and performance aspects of this shared-output buffered architecture will be discussed below.

4 Operating a shared-output buffered switch.

Figure 3 "Delay vs. Load characteristic of shared-output buffered switch’ shows one of the most important parameters of a switch: because there is competition for outputs, which is solved by queuing, the more packets there are being queued, the longer a single packet will take to traverse the switch. As the load increases, there will be more competition, and thus longer delays. At some point the internal queue storage is exhausted and packets will be lost, or the delay saturates to infinity. Not allowing loads that cause a very steep or long delay is tantamount to limiting the throughput of the switch. Saturation of the throughput of these switches is dependent on the ratio of the mean available output queue storage per output and the average packet length. It should be noted that the effect of sharing the output buffer among several outputs becomes noticeable only when there are at least 8 outputs that share. It is necessary to always keep the operating point of the switch below the knee-point where the normalized delay increases quickly to infinity with only a small increase in the offered load. Note that the actual average packet size is very difficult to estimate, and often changes quickly. This means that the
actual saturation point changes dynamically, depending on the offered traffic characteristic. It is therefore advisable to maintain a sufficient margin between the saturation point and the operating point. In fact, in order to achieve lossless operation, a local flow-control mechanism is needed that activates relatively low-cost queue storage on the input adapters when the switch momentarily reaches saturation. Note that this mechanism is not suited to operate the switch for longer periods of time at a point closer to the saturation point.

5 References

An Introduction To Message Passing Paradigms

David W. Walker
Oak Ridge National Laboratory, Oak Ridge, U.S.A.

Abstract

This paper discusses some of the most important message passing approaches used on high performance computers, and describes their central features. Occam and Fortran M are coordination languages in which processes communicate through typed channels. Split-C and Linda provide support for shared memory programming. In Split-C all processes can access a global address space through global pointers. The central feature of Linda is an associative shared address space called “tuple space.” PVM and MPI are examples of message passing libraries that are typically used with sequential languages such as C and Fortran. MPI provides support for modular application libraries through the communicator abstraction. PVM features extensive job and process control functionality which has resulted in it being widely used on networks of workstations.

Keywords: Parallel computing; message passing; occam; Fortran M; Linda; Split-C;PVM;MPI

1 Introduction

This paper gives an introduction to some of the more widely-used and interesting approaches to the message passing style of programming high performance computers, in particular distributed memory computers. It is not my intent to give a complete review of the message passing paradigm, nor to describe in detail the features of any particular approach. Rather, this paper is an eclectic look at a few of the languages, extensions, and message passing libraries that are widely-used and/or have been influential in advancing parallel programming and the programming of parallel computers. This paper should be useful to novices wanting to get a basic understanding of some of the options open to them for using parallel computers. Wherever possible references are given to more detailed information available in the literature or electronically should readers wish to delve more deeply.

The rest of this paper is organized as follows. Section 2 presents an overview of message passing, and places it in the broader context of parallel programming paradigms. In Section 3, coordination languages are discussed, and the main features of Occam and Fortran M are described. Linda and Split-C, described in Section 4, strictly speaking are based on a shared memory approach, but they are included here on the basis of their strong message passing flavor. Two message passing libraries, PVM and MPI, are discussed in Section 5.

2 The Message Passing Approach

In the past decade the concept of message passing has become closely associated with the efficient use of parallel multicomputers and with distributed computing. However, as Hoare relates in his very readable article on occam and the transputer [25], the concept of message passing was also strongly motivated by its use in the mid-1960s in the design of operating systems. The idea of communicating sequential processes as a model for parallel execution was developed by Hoare [24] in the 1970s, and is the basis of the message passing paradigm. This paradigm assumes a distributed process memory model, i.e., each process has its own local address space. Processes cooperate to perform a task by independently computing with their local data and communicating data between processes by explicitly exchanging messages. The message passing
approach is particularly well-suited to computers with physically distributed memory since there is a good match between the distributed memory model and the distributed hardware. However, message passing can be used on shared memory and sequential computers, and, indeed, can be used as the basis for the development of portable and efficient programs on all these architectures. An alternative approach is based on the shared process memory model, in which all processes have access to the same global address space. As might be expected, this approach works well on shared memory architectures, but may also be supported in software on distributed memory computers. An example is Orca which is a language that supports shared objects [9]. Other alternatives to the message passing approach include data parallel extensions to sequential languages, such as High Performance Fortran [27], and implicit parallel languages employing functional parallelism, such as Sisal [16].

There are two main ways of writing a parallel message passing program. The first is through a coordination language such as Fortran M [20] or occam [28]. These are specialized languages for specifying concurrency, communication and synchronization, and will be discussed in greater detail in Section 3. The second way of performing message passing is with calls to a message passing library from within a sequential program. This has proved to be a very popular way of writing concurrent applications since it is expressive, closely models the functionality of the parallel hardware, and permits explicit management of the memory hierarchy. A number of different message passing libraries have been developed over the past decade in addition to the hardware specific message passing libraries provided by different parallel computer vendors. Express is a commercial parallel computing environment produced by Parasoft Inc. [30], and is based on the CrOS III message passing library [21]. Other message passing libraries include PARMACS [11], which is based on p4 [10], and Zipcode [32]. The special issue of Parallel Computing on message passing include more detailed articles on these, and other, message passing libraries [1].

3 Coordination Languages

Coordination languages are languages with explicit support for expressing parallelism, communication, and synchronization. In this section two examples, occam and Fortran M, will be considered.

3.1 Occam

Ocam was designed specifically for programming transputers, indeed, it is often referred to as the assembly language of the transputer. Occam is based on Hoare’s model of communicating sequential processes [24] which allows programs to be analyzed mathematically to check for program correctness. In occam, parallelism is expressed through the PAR construct. Thus, the code fragment

```
PAR
  process 1
  process 2
  process 3
```

indicates that the three processes may be executed in parallel. Similarly, the SEQ construct is used to specify the sequential execution of a set of processes. There are three primitive processes in occam, namely, input to a channel, output from a channel, and assignment. More complex processes can be built from these primitives by defining procedures. In occam, messages
are communicated through uni-directional, typed channels. Each channel has one sender process and one receiver process. A channel is said to be typed if the datatypes of the variables that may be communicated by the channel are predefined. The following occam code illustrates a simple use of a typed channel.

```occam
CHAN OF INT chan :
PAR
  chan ! 2
INT x, z :
SEQ
  chan ? x
  z := x
```

The above program illustrates the use of the three primitive processes, as well as the PAR and SEQ constructs. The first line of code declares a channel, `chan`, that will be used to communicate an integer. The PAR construct refers to two processes. One process outputs the integer 2 on channel `chan (chan ! 2)`. The second process is a compound process beginning with the SEQ and consists of two primitive subprocesses executed sequentially. The first inputs the data from channel `chan` into the variable `x (chan ? x)`, and the second copies the value of `x` to variable `z (z := x)`. Note that `x` and `z` are declared immediately before the process that uses them, and that their scope is limited to just that process.

In occam, communication is synchronous. This means that the sending process must output to a channel and the receiving process must input from that channel in order for the communication to occur. If either the sending or receiving process is not yet ready to participate in the communication then the other process will wait. In incorrectly designed programs deadlock may occur. Deadlock arises in a parallel program if one or more processes are waiting for an event that will never occur, and is a common programming bug. The following program gives an example of deadlock.

```occam
CHAN OF INT chan1, chan2 :
PAR
  INT x :
SEQ
  chan1 ! 2
  chan2 ? x
INT z :
SEQ
  chan2 ! 4
  chan1 ? z
```

Here, one process outputs the value 2 on channel `chan1`, and another outputs the value 4 on channel `chan2`. Both processes then wait indefinitely because neither can continue until the other inputs to their respective channels. Thus, deadlock occurs. In this case, deadlock can be avoided simply by reversing the order of the input and output processes in either (but not both!) of the parallel processes.

Communication is said to be non-deterministic if the receive on the receiving process may be matched by any one of several potential send operations on other processes. Non-deterministic communication is often used to improve the performance of a parallel program by ensuring that the receiving process is supplied with enough data from other processes to keep it busy. However, non-determinism may be introduced unintentionally into a program and is the second most
common type of bug in parallel programming. A program with this sort of bug may fail intermit-
tently since its behavior depends on the synchronization between processes. Non-determinism
may be introduced into an occam program through the ALT construct. An ALT construct moni-
tors several channels, each of which is associated with a process. Only one of these processes is
executed, and that is the one associated with the channel that first produces an input. Consider
the following code fragment.

```occam
CHAN OF INT chan1, chan2 :
INT x, y :
ALT
  chan1 ? x
  SEQ
    y := 2*x + 1
    chan3 ! y
  chan2 ? x
  SEQ
    y := 2*x
    chan3 ! y
```

Here a process waits for input on either chan1 or chan2. When it receives input on either
channel it performs a simple computation and outputs the result on chan3. This code fragment
does not show the processes that produce the input or consume the output. If the process receives
input x first on chan1 it computes 2x + 1 and outputs the results on chan3. If it receives input
first on chan2 it will output 2x on chan3. Thus, there are two possible outcomes, and it is not
possible to determine a priori which will actually occur.

In addition to the PAR, SEQ, and ALT constructs occam also provides IF and WHILE con-
structs for controlling program flow. All of these constructs, except WHILE, may be replicated.
Replicated SEQ and PAR constructs create sets of similar processes that execute sequentially or
in parallel. A replicated PAR may be used for the single-program, multiple-data (SPMD) style of
programming. A replicated ALT can be used to monitor an array of channels, and a replicated IF
can be used to create a conditional with similar clauses. Occam also provides a mechanism for
placing processes on physical processors which may improve the performance of a program, but
does not affect its correctness. Another important and useful feature is occam’s use of protocols
to allow channels to carry data of mixed types. Protocols can also be used for communica-
ting fixed and variable length arrays, and for allowing a single channel to carry messages of differing
format.

It is hoped that the above gives a good idea of the main features of the occam language
and how they are used. For further information the interested reader is referred to the excellent
(and short) introduction by Pountain and May [31], and to the more advanced text by Jones and
Goldsmith [26] which also contains a good bibliography. The occam web page contains pointers
to information on occam compilers, documentation, and projects [2].

3.2 Fortran M

Fortran M was developed by Ian Foster of Argonne National Laboratory and Mani Chandy of
the California Institute of Technology [20], and consists of Fortran 77 plus a set of extensions for
process creation and control, and communication. Fortran M is broadly similar to occam in that it
provides constructs for creating parallel processes, communicates data through uni-directional,
typed channels, and supports the mapping of processes to processors. However, there are im-

portant differences. One important difference is that communication in Fortran M is not syn-
chronous because the send may complete before the matching receive operation is started at the
other end of the channel. The second important difference is that channels are specified as links
between *inports* and *outports*. A process sends data through an outport and receives data through
an inport. Furthermore, ports may be communicated giving rise to dynamic channels. Fortran M
does not have any equivalent of the *SEQ*, *WHILE*, and *IF* constructs as these are subsumed within
the Fortran language. Fortran M introduces non-determinism through the *MERGER* and *PROBE*
constructs, which replace occam’s *ALT* construct.

The following program illustrates the basic message passing capabilities of Fortran M.

```fortran
program swap1
  IMPORT (integer) portin(2)
  OUTPORT (integer) portout(2)
  CHANNEL (in=portin(1), out=portout(2))
  CHANNEL (in=portin(2), out=portout(1))
  PROCESSES
    PROCESSCALL swap (portin(1), portout(1))
    PROCESSCALL swap (portin(2), portout(2))
  ENDPROCESSES
end
```

This program declares two inports and two outports capable of receiving and sending an integer.
Two channels are then established, each connecting an inport to an outport. The section of code
beginning with “*PROCESSES*” and ending with “*ENDPROCESSES*” constitutes a process block, and
is similar to the *PAR* construct in occam. The two processes in the process block are executed
concurrently, with each process calling *swap* with different inport and outport arguments. The
*swap* processes exchange an integer value, *p*. Note that deadlock will not occur here because the
send operation can complete before the matching receive operation is initiated.

In Fortran M the *MERGER* construct can be used to connect multiple outports to a single
inport. This can be used to introduce non-determinism into a parallel program since data from
only one of the outports will be received at the inport. The *MERGER* construct is equivalent to
a replicated *ALT* in occam. The following Fortran M code fragment connects three producer
processes to one consumer process.

```fortran
INPORT (integer) portin
OUTPORT (integer) portout(3)
MERGER (in=portin, out=pout(:))
PROCESSES
  call consumer (pin)
  PROCESSDO i=1,3
    PROCESSCALL producer (pout(i))
  ENDPROCESSDO
```
In this example the `MERGER` statement connects the three elements of the array of outports `portout` to the inport `portin`. Within the process block the subroutine `consumer` is executed on the current process, and the `PROCESSDO` loop creates three `producer` processes. The `consumer` routine is passed the inport of the merger and each `producer` process is passed one of the outports.

Nondeterminacy may also be introduced into a Fortran M program through the `PROBE` construct which is used to check if a message is waiting to be received on an inport. `PROBE` returns a logical variable to indicate if a message is pending on a specified inport, and is typically used to allow a process to do useful work while intermittently checking if a message is ready for receipt, thereby avoiding idle time.

A useful feature of Fortran M is the ability to communicate an inport or outport from one process to another. By communicating a port the processes connected by a channel can be changed. Thus, instead of statically connecting two fixed processes channels may be dynamic. Dynamic channels are often used when a process wants another process to return data to it. In such cases the process creates a channel and then sends the channel’s outport to another process. That process then has the ability to send data back to the original process. This approach is useful for dealing with “migrating computations” in which a task may move between processes before completing, and then sends back the results of the work to the process that originated the task.

Fortran M provides mechanisms for process placement. As in occam, this affects the performance, but not the correctness, of programs. Process placement is based on the concept of a `virtual process array` which is declared in the Fortran M program. Fortran M provides annotations that can be used to specify where in the virtual process array a process is to run, and for subdividing the virtual process array so that different processes run on different “subcomputers.”

For further details of the Fortran M language readers are referred to the paper by Foster and Chandy [20], and to the Fortran M manual [18]. Foster has also written a book that discusses Fortran M as well as other approaches to parallel programming, such as CC++ and High Performance Fortran, that are not covered in this paper [19]. All of these publications are available from the Fortran M web page at Argonne National Laboratory [3]. If you want to try out Fortran M for yourself the necessary software can also be obtained from the web page.

## 4 Shared Memory Paradigms

In this section we discuss two alternatives to the message passing paradigm, Linda and Split-C. These provide support for shared memory, and were designed with distributed memory implementations in mind.

### 4.1 Linda

Linda was developed by Gelertner and colleagues, and is a set of simple extensions to a conventional programming language such as C or Fortran. Linda is based on an associative shared memory known as “tuple space.” A tuple is an ordered sequence of data. For example the tuple

\[ (\text{'array x'}, 5, [1 4 2 7 9]) \]

is a tuple consisting of a string, an integer, and an array of integers. In Linda processes only interact through tuple space, and routines are provided for placing tuples into, and extracting tuples from, tuple space. Tuples may be created and placed in tuple space using the `out` and `eval` operations. Tuples may be read or removed from tuple space using the `rd` and `in` operations. Linda’s extensions consist of just these four operations making it very simple, but at the same time very expressive.
The **out** operation evaluates the tuple fields sequentially on the calling process, and places the resulting tuple into tuple space. For example

```plaintext
out ('function', i, f(i))
out ('common data', /params/)
out ('array section', i, j, x(i:j)).
```

For the first of these the calling process evaluates the function \( f \) for argument \( i \), making this the third component of the tuple, and places the resulting tuple into tuple space. The second example places a tuple into tuple space consisting of the string `'common data'` and contents of the common block `params`. The third example shows how to place an array section into tuple space consisting of a string, two integers (with \( i \geq j \)), and a section of the array \( x \) starting with \( x(i) \) and going up to \( x(j) \). It should be noted that **out** is similar to a send operation, but is more general — instead of sending data from one process to another data are sent from a process to tuple space, where it can be sequently retrieved by another process. Also Linda supports self-describing messages in a very natural way.

The **eval** operation can be used as a general way of spawning processes. It evaluates the tuple fields in parallel and then places the resulting tuple in tuple space. For example the code fragment

```plaintext
do i=/1/,/1/0/0
eval(/'function/', i, f(i))
end do
```

may cause 100 new processes to be created, each of which handles one of the tuples, placing it in tuple space after performing the function evaluation. The **eval** operation is nonblocking on the calling process so the calling process continues without waiting for the **eval** operation to complete. **eval** is typically called by a host process to spawn a set of worker processes which return their results to tuple space, allowing them to be retrieved by the host which can then process them further and/or output results.

There are two operations for retrieving tuples from tuple space. The **in** operation finds a tuple that associatively matches a given template and removes it from tuple space. The **rd** operation is similar to **in** but does not remove the tuple from tuple space. For example the tuple placed into tuple space with

```plaintext
out ('message', i, n, data(1:n))
```

can be retrieved with

```plaintext
in ('message', i, ?size, ?mydata(1:n))
```

This will match all tuples in tuple space with four fields whose datatypes match, and whose first two fields are `'message'` and \( i \). The question marks in front of the last two fields indicate that the values of these two fields do not matter in establishing a match. When a matching tuple is retrieved the variable \( size \) takes the value of the third field, and the array `mydata` receives the values of the array section in the fourth field. If more than one matching tuple is found one is selected at random. If no matching tuple is found the process waits until one is available.

To illustrate how Linda is used consider the following example in Linda Fortran which integrates the function \( \sin x \) from 0 to \( \pi \).

```fortran
program integrate
parameter (npts=100, pi=3.141592654)
deltax = pi/npts
do i=2,npts
   x = (i-0.5)*deltax
eval(sin(x))
end do
```

7
In this example the real axis is divided into \texttt{npts} equally-sized intervals, and the integral within each integral is approximated by the value of \(\sin x\) evaluated at the mid-point of the interval multiplied by the width of the interval. At the start of the program a single process exists which we shall call the host process. The \texttt{eval} operation in the first loop causes \texttt{npts-1} worker processes to be created, each of which evaluates the integrand at the mid-point of one interval. The host process evaluates the integrand at the mid-point of the first interval. Having evaluated their respective integrands the worker processes place their results in tuple space. In the second loop the host process retrieves the results from tuple space, sums them together, and outputs the approximate integral. Of course, this is trivial example since the integrand is so simple. However, the integrand can be arbitrarily complicated and perhaps involve a lengthy computation. For such problems it makes sense to use a parallel computer.

It should be noted that Linda is capable of simulating message passing by libraries such as PVM and MPI, discussed in Section 5. For example, if a tuple is placed in tuple space as follows
\[
\text{out('message', myrank, dest, tag, 'real', n, data(1:n))}
\]
and is retrieved thus
\[
\text{in('message', source, myrank, tag, 'real', ?n, ?data(1:n))}
\]
this is similar in MPI to a standard blocking send matched by a blocking receive that communicates \(n\) values in a real array from the process with rank \texttt{source} to that with rank \texttt{dest}. \texttt{myrank} is the rank of the calling process in each case.

Linda is well-suited for handling dynamic load balancing by treating tuple space as a “bag of tasks.” Tasks to be performed can be placed into tuple space with \texttt{eval}, and on completion of a task the results are put back into tuple space.

An important distinction between Linda and approaches using message passing is that there is only a loose coupling between processes — processes interact only with tuple space and not directly with each other. Thus, a process that consumes data in tuple space does not have to even exist at the time the data is placed into tuple space.

Linda is commercially supported by Scientific Computing Associates, Inc. Gelertner and colleagues have written several papers on Linda, for example [12, 13, 14]. These papers and lots more information about Linda are available from the Linda web page at Yale University [4]. There is also an active Linda group at York University in the United Kingdom [5].

### 4.2 Split-C

Split-C is being developed by David Culler and colleagues at the University of California, Berkeley. Split-C is a set of parallel extensions to the C programming language that provide mechanisms for supporting shared memory. Split-C is based on the single-program, multiple-data (SPMD) model, i.e., a fixed number of processes all execute the same executable code. In Split-C, processes can access a global address through global pointers. We may regard this address space as two-dimensional with processes running in one direction and local memory in the other.
In addition each process can access its own local address space using standard pointers. Derefer-
encing a global pointer may involve communication and be more expensive than dereferencing a
global pointer. Hence, in the interests of performance, global pointers should be used sparingly.
The following lines of Split-C illustrate some of the uses of global pointers.

```c
int *global gptr;
int xval, proc, *lptr;
xval = &gptr;
gptr = toglobal(MYPROC, lptr);
lptr = tolocal(gptr);
proc = topproc(gptr);
```

The first of the above lines shows how to declare a global pointer. The second line dereferences a
global pointer into a local variable. The next three lines make use of some Split-C functions.
toglobal converts a local pointer to a global pointer. MYPROC is a special integer that uniquely
numbers a process in the range 0 to PROCS-1, where PROCS is another special integer giving the
number of processes. The functions tolocal and topproc destructure a global pointer into its
local pointer and process number parts. The following routines shows how a global pointer could
be used to broadcast n integers from a root process to all processes.

```c
all_bcast_int(int root, int *val, int n)
{
    int i;
    int *global = toglobal(root, val);
    barrier();
    for(i=0; i<n; i++) *(val++) = *(gptr++);
    barrier();
}
```

The toglobal function is used to create a global pointer that references the start of the data on
the root process. A barrier is then performed to make sure that the data to be broadcast from the
root process is ready. Then each process dereferences the global pointer so the value it points to
is stored at the address pointed to by the local pointer, val. By incrementing the local and global
pointers each of the n elements are broadcast. A second barrier then has to be performed to make
sure that the root process does not overwrite the broadcast data before all the other processes
have received it. This is almost certainly not a very efficient way to perform a broadcast because
each time the global pointer is dereferenced data must be communicated from the root to all
other processes. It should be noted that when arithmetic is performed on a global pointer another
object on the same process is accessed. It is an error to do arithmetic on a global pointer outside
of the local address space.

Spread pointers are global pointers that traverse the two-dimensional global address space
in the process direction, so that successive objects are on different processes. Spread pointers
wrap around in the process direction so if sptr is a spread pointer, sptr+PROCS points to the next
object on the same process, and sptr+1 points to the object on the next process. The following
example shows how to use a spread pointer to sum a set of numbers distributed one to a process.

```c
/* include files should go here */
splitc_main(){
    int *spread sptr;
    int *lptr, i, sum=0;
    sptr = all_spread_malloc(PROCS, sizeof(int));
    lptr = tolocal(sptr);
```
In this example the routine `all_spread_malloc` is called to create a spread pointer to an integer over the processes. Each process extracts the local part of the spread pointer and stores its process number at that location. Then the macro `on_one` is used to sum the values stored at the spread pointer locations. Note the use of barriers to ensure proper synchronization.

Split-C also provides spread arrays in which one or more dimensions are spread over the processes. By convention, the array dimensions that are distributed lie to the left of the spreader `::`. For example,

```c
int x[10*PROCS]::[10];
```

declares an array `x` whose first dimension is cyclically distributed over the processes, so `x[i][j]` refers to the `j`th element on process `i mod PROCS`. This is equivalent to a vector of `100*PROCS` elements block cyclically distributed with a block size of 10. The summation example shown above can be rewritten to use a spread array as follows.

```c
static int sarray[PROCS]::;
/* include files should go here */
splitc_main(){
  int i, sum=0;
  sarray[MYPROC] = MYPROC;
  barrier();
  on_one{
    for(i=0;i<PROCS;++i) sum += sarray[i];
    printf("\nThe sum is %i\n",sum);
  }
  barrier();
  all_spread_free(sptr);
  exit(0);
}
```

In the preceding Split-C examples global objects have been accessed one at a time. This is likely to be inefficient because the latency associated with the communication is paid for each access. Split-C therefore provides bulk assignment routines for copying to or from arrays of global objects. Thus, the broadcast example above could be implemented by having each process call the routine `bulk_read` to read the data from the root process into local memory in one operation.

In addition to support for a global address space, the second important feature of Split-C is split phase assignment. Split phase assignment allows access to a global object to be overlapped with other useful work. The assignment initiates access to the global object, but the access is guaranteed to have completed only after a subsequent call to the routine `sync()`. The “get” form of split phase assignment places the content of a global reference into a local one, and the “put” form places the contents of a local reference into a global one. The split phase operator is `::=`.
The get form of the broadcast example may be written as follows,

```c
void all_bcast_int(int root, int *val, int n)
{
    int i;
    int *global gp = toglobal(root, val);
    barrier();
    for(i=0; i<n; i++) *(val++) = *(gp++);
    sync();
    barrier();
}
```

In this version of the example we do not have to wait for one assignment to finish before starting the next, so the use of split phase assignment should result in more efficient code. There are also routines for performing bulk split phase assignment.

When storing into a global location using a standard assignment an acknowledgement is returned to the initiating process when the store has completed. Split-C provides another form of assignment, called store assignment, that does not perform this acknowledgement, and is hence faster. A subsequent call to the routine `all_store_sync` synchronizes processes and waits for all the stores to complete. The store assignment operator is `:-`. The following example shows how to use store assignment in the put form of a broadcast in which the root stores the data on each process.

```c
void all_bcast_int(int root, int *val, int n)
{
    int *global gp;
    int i, ip;
    barrier();
    on_proc(root){
        for(ip=0; ip<PROCS; ip++) {
            gp = toglobal(ip, val);
            for (i=0; i<n; i++) *(gp++) = *(val++);
        }
    }
    all_store_sync();
}
```

In this example the root loops over all processes, creates a global pointer into each process’ memory, and then does a store assignment to that location. There is also a bulk store assignment routine for the store assignment of arrays.

Only a brief introduction into Split-C has been given here. Split-C also contains a number of macros and auxiliary library routines for performing tasks such as broadcast and reduction. For further details the reader is referred to the tutorial introduction by Culler et al. [15]. The software for implementing Split-C, as well as other information, is available from the world wide web [6].

## 5 Message Passing Libraries

This section describes two message passing libraries that have emerged as potential standards: PVM and MPI. These libraries are designed for use with a sequential programming language such as C or Fortran. Central to each library is a set of routines for performing point-to-point
communication between pairs of processes. Additional routines perform collective communication operations, such as broadcasts and reductions, and process and job management. A recent special issue of the journal Parallel Computing on the topic of message passing contains articles on several widely-used message passing environments [1].

5.1 Parallel Virtual Machine (PVM)

The earliest version of PVM was developed by Vaidy Sunderam in 1989 and 1990 while visiting Oak Ridge National Laboratory (ORNL) [33]. Later versions of PVM were developed by a team of researchers at ORNL, the University of Tennessee, Knoxville, Emory University and Carnegie Mellon University. PVM enables a collection of different computing systems to be viewed as a single parallel machine. In particular, it has been very widely used to connect networks of workstations (NOWs) together to execute parallel programs. The component computers in the parallel virtual machine need not be all of the same type, and PVM will transparently handle any necessary data conversions when messages are passed between machines with different internal data representations. Currently C and Fortran interfaces to PVM exist.

PVM is more than just a message passing library — it is a complete environment for heterogeneous parallel computing consisting of four main parts.

1. The PVM daemon. This runs on every host computer in the parallel virtual machine. It controls PVM resources on each host and mediates interactions with other hosts.
2. The PVM console. This is a user interface that allows the user to interactively configure the parallel virtual machine.
3. The PVM group server. This manages process groups.
4. The PVM library. The set of routines for performing message passing, and managing tasks and process groups.

The remainder of this subsection will focus on the PVM library.

The most natural way to program with PVM is to use a host-worker model of computation in which a single host process is responsible for doing I/O and for spawning a set of worker processes that concurrently perform most of the computation. The host and worker processes are each uniquely identified by an integer known as the task ID.

In PVM, point-to-point messages are typed, and tagged. A typed message is one in which the message passing system is aware of the datatypes of the components making up a message. PVM is capable of sending messages of mixed datatypes. A tagged message is one that has an integer “tag” associated with it. When a message is sent the destination is specified by the task ID of the receiving process. Similarly, when a message is received the source of the message is identified by its task ID. Message selectivity is by source task ID and tag. Thus, when a message is received the source task ID and tag associated with the message must match those specified in the argument list of the receive routine, unless either is wildcarded. If the receive routine is passed a value of $-1$ for the source task ID then this criterion will be ignored in message selection and we say that it is wildcarded. The tag may be wildcarded in a similar way.

Message passing is done by calls to routines in the message passing library and makes use of system controlled buffers. Typically, sending a message takes place in three phases. First the system send buffer is cleared and prepared for use. Next the message is packed into the message buffer. Lastly the message is sent. Receiving a message usually takes place in two phases. First the message is received into the system receive buffer and then it is unpacked into the application space. The following code fragment shows how one processes sends a message consisting of one integer followed by two floats to another process.
tid = pvm_mytid();
if (tid == source) {
    bufid = pvm_initsend(PvmDataDefault);
    info = pvm_packint(&i1, 1, 1);
    info = pvm_packfloat(vec1, 2, 1);
    info = pvm_send(dest, tag);
}
elseif (tid == dest) {
    bufid = pvm_recv(source, tag);
    info = pvm_upkint(&i2, 1, 1);
    info = pvm_upkfloat(vec2, 2, 1);
}

In this example each process calls the routine pvm_mytid() to determine its task ID. It is assumed that the values of source and dest are distinct valid task IDs for the set of processes executing the code. The source process initializes the send buffer, and packs an integer (i1) and two floats (the first two elements of vec1) into it. Finally the source process sends the data to the destination process, dest. The destination process waits to receive the message with the specified tag value from the source process. Once it has been received it unpacks the integer into variable i2 and the two floats into the first two elements of vec2.

The receive routine pvm_recv is blocking, i.e., if no message satisfying the selection criteria is available it will wait until one is available. The send routine pvm_send is nonblocking in the sense that a call to pvm_send initiates the send operation and then returns. PVM provides a receive routine pvm_nrecv that is nonblocking in the sense that if a message satisfying the selection criteria has not arrived when it is called it returns a value of zero, otherwise it receives the message into a new receive buffer and returns the strictly positive buffer ID. PVM also provides a receive routine that times out if a suitable message is not received within a specified time. The packing and unpacking of data is useful when messages of mixed datatype need to be communicated, but is rather inconvenient for messages of a single datatype. In such case the routines pvm_psend and pvm_precv may be used. For these routines the data buffer to send, or receive into, is specified in the argument list, together with the datatype. The routine pvm_psend is blocking, i.e., it will not return until it is safe to reuse the data buffer. The routine pvm_precv is also blocking, just like pvm_recv. Finally, the routine pvm_probe can be used to check if a message with a specified tag from a specified source is ready for receipt without actually receiving it.

In addition, to point-to-point communication and related routines PVM also includes a small set of collective communication routines. These routines involve coordinated communication within groups of processes. PVM provides mechanisms that allow processes to asynchronously join or leave a process group, and groups are identified by a user-supplied string. The three collective communication operations are barrier synchronization, broadcast, and reduction. The barrier routine ensures that no process in the group exits the routine until they all have entered it. The broadcast routine sends data from one process in the process group to all processes in the group. The reduction routine combines the values provided in the input buffer of each process in the group using a specified function. Thus, if $D_i$ is the data in the $i$th process in the group, and $\oplus$ is the combining function, then the following quantity is evaluated,

$$\mathcal{D} = D_0 \oplus D_1 \oplus D_2 \oplus \cdots \oplus D_{n-1}$$

where $n$ is the size of the process group. The result $\mathcal{D}$ is returned to a specified root process. The
combining function is supplied by the user, although PVM provides some pre-defined functions, such as summation.

Further details of PVM can be best obtained from the PVM users’ guide and tutorial which is available as a book [22]. The PVM web page provides further documentation, recent news on PVM, research projects involving PVM, and pointers to other PVM resources [7]. The software for implementing PVM in a variety of computing environments is also available from the web page.

### 5.2 Message Passing Interface (MPI)

MPI is a proposed standard message passing interface. The design of MPI was a collective effort involving researchers in the United States and Europe from many organizations and institutions. MPI includes point-to-point and collective communication routines, as well as support for process groups, and application topologies.

In MPI there is currently no mechanism for creating processes, and an MPI program is parallel *ab initio*, i.e., there is a fixed number of processes from the start to the end of an application program. All processes are members of at least one process group. Initially all processes are members of the same group, and a number of routines are provided that allow the user to create (and destroy) new subgroups. Within a group each process is assigned a unique rank in the range 0 to $n - 1$, where $n$ is the number of processes in the group. This rank is used to identify a process, and, in particular, is used to specify the source and destination processes in a point-to-point communication operation, and the root process in certain collective communication operations. As in PVM, message selectivity in point-to-point communication is by source process and message tag, each of which may be wildcarded to indicate that any valid value is acceptable.

The key innovative ideas in MPI are the communicator abstraction and general, or derived, datatypes. These will be discussed before describing the communication routines in more detail. Communicators provide support for the design of safe, modular software libraries. Here “safe” means that messages intended for a particular receive routine in an application will not be incorrectly intercepted by another receive routine. Thus, communicators are a powerful mechanism for avoiding unintentional non-determinism in message passing. This is a particular problem when using third-party software libraries that perform message passing. The point here is that the application developer has no way of knowing if the tag, group, and rank completely disambiguate the message traffic of different libraries and the rest of the application. Communicator arguments are passed to all MPI message passing routines, and a message can be communicated only if the communicator arguments passed to the send and receive routines match. Thus, in effect communicators provide an additional criterion for message selection, and hence permit the construction of independent tag spaces.

If communicators are not used to disambiguate message traffic there are two ways in which a call to a library routine can lead to unintended behavior. In the first case the processes enter a library routine synchronously when a send has been initiated for which the matching receive is not posted until after the library call. In this case the message may be incorrectly received in the library routine. The second possibility arises when different processes enter a library routine asynchronously, as shown in the example in Figure 1, resulting in nondeterministic behavior. If the program behaves correctly processes 0 and 1 each receive a message from process 2, using a wildcarded selection criterion to indicate that they are prepared to receive a message from any process. The three processes then pass data around in a ring within the library routine. If separate
Communicators are not used for the communication inside and outside of the library routine this program may intermittently fail. Suppose we delay the sending of the second message sent by process 2, for example, by inserting some computation, as shown in Figure 2. In this case the wildcarded receive in process 0 is satisfied by a message sent from process 1, rather than from process 2, and deadlock results. By supplying a different communicator to the library routine we can ensure that the program is executed correctly, regardless of when the processes enter the library routine.

![Figure 1: Use of communicators. Time increases down the page. Numbers in parentheses indicate the process to which data are being sent or received. The gray shaded area represents the library routine call. In this case the program behaves as intended. Note that the second message sent by process 2 is received by process 0, and that the message sent by process 0 is received by process 2.](image1)

![Figure 2: Unintended behavior of program. In this case the message from process 2 to process 0 is never received, and deadlock results.](image2)

Communicators are opaque objects, which means they can only be manipulated using MPI routines. The key point about communicators is that when a communicator is created by an MPI routine it is guaranteed to be unique. Thus it is possible to create a communicator and pass it to a software library for use in all that library’s message passing. Provided that communicator is not used for any message passing outside of the library, the library’s messages and those of the rest of the application cannot be confused.

Communicators have a number of attributes. The group attribute identifies the process group relative to which process ranks are interpreted, and/or which identifies the process group involved in a collective communication operation. Communicators also have a topology attribute which gives the topology of the process group. Topologies are discussed below. In addition, users may associate attributes with communicators through a mechanism known as caching.
All point-to-point message passing routines in MPI take as an argument the datatype of the data communicated. In the simplest case this will be a primitive datatype, such as an integer or floating point number. However, MPI provides a number of routines for creating more general datatypes, and thereby supports the communication of array sections and structures involving combinations of primitive datatypes.

In many applications the processes are arranged with a particular topology, such as a two- or three-dimensional grid. MPI provides support for general application topologies that are specified by a graph in which processes that communicate a significant amount are connected by an arc. If the application topology is an $n$-dimensional Cartesian grid then this generality is not needed, so as a convenience MPI provides explicit support for such topologies. For a Cartesian grid periodic or nonperiodic boundary conditions may apply in any specified grid dimension. In MPI, a group either has a Cartesian or graph topology, or no topology. In addition to providing routines for translating between process rank and location in the topology, MPI also:

1. allows knowledge of the application topology to be exploited in order to efficiently assign processes to physical processors,
2. provides a routine for partitioning a Cartesian grid into hyperplane groups by removing a specified set of dimensions,
3. provides support for shifting data along a specified dimension of a Cartesian grid.

By dividing a Cartesian grid into hyperplane groups it is possible to perform collective communication operations within these groups. In particular, if all but one dimension is removed a set of one-dimensional subgroups is formed, and it is possible, for example, to perform a multicast in the corresponding direction.

A set of routines that supports point-to-point communication between pairs of processes forms the core of MPI routines for sending and receiving blocking and nonblocking messages are provided. A blocking send does not return until it is safe for the application to alter the message buffer on the sending process without corrupting or changing the message sent. A nonblocking send may return while the message buffer on the sending process is still volatile, and it should not be changed until it is guaranteed that this will not corrupt the message. This may be done by either calling a routine that blocks until the message buffer may be safely reused, or by calling a routine that performs a nonblocking check on the message status. A blocking receive suspends execution on the receiving process until the incoming message has been placed in the specified application buffer. A nonblocking receive may return before the message has been received into the specified application buffer, and a subsequent call must be made to ensure that this has occurred before the application uses the data in the message.

In MPI a message may be sent in one of four communication modes, which approximately correspond to the most common protocols used for point-to-point communication. In ready mode a message may be sent only if a corresponding receive has been initiated. In standard mode a message may be sent regardless of whether a corresponding receive has been initiated. MPI includes a synchronous mode which is the same as the standard mode, except that the send operation will not complete until a corresponding receive has been initiated on the destination process. Finally, there is a buffered mode. To use buffered mode the user must first supply a buffer and associate it with a communicator. When a subsequent send is performed using that communicator MPI may use the associated buffer to buffer the message. A buffered send may be performed regardless of whether a corresponding receive has been initiated. In PVM message buffering is provided by the system, but MPI does not mandate that an implementation provide message buffering. Buffered mode provides a way of making MPI buffer messages, and is useful when converting a program from PVM to MPI.
In addition, MPI provides routines that send to one process while receiving from another. Different versions are provided for when the send and receive buffers are distinct, and for when they are the same. The send/receive operation is blocking, so does not return until the send buffer is ready for reuse, and the incoming message has been received.

MPI includes a rich set of collective communication routines that perform coordinated communication among a group of processes. The process group is that associated with the communicator that is passed into the routine. MPI’s collective communication routines can be divided into two groups: data movement routines and global computation routines. There are five types of data movement routine: broadcast, scatter, gather, all-gather, and all-to-all. These are illustrated in Fig. 3.

There are two global computation routines in MPI: reduce and scan. The MPI reduction operation is similar in functionality to that provided by PVM. Different versions of the reduction routine are provided depending on whether the results are made available to all processes in the group, just one process, or are scattered cyclicly across the group. The scan routines perform a parallel prefix with respect to a user-specified operation on data distributed across a specified group. If \( D_i \) is the data item on the process with rank \( i \), then on completion the output buffer of this process contains the result of combining the values from the processes with rank \( 0, 1, \ldots, i \), i.e.,

\[
D_i = D_0 \oplus D_1 \oplus D_2 \oplus \cdots \oplus D_i
\]

Two versions of the MPI specification exist. One is dated May 5, 1994 (version 1.0), and the other June 12, 1995 (version 1.1). The latter document incorporates corrections and clarifications to the former, but the two do not differ in any substantial way. At the time of writing version 1.1 is only available electronically [17]. The book on using MPI by Gropp, Lusk and Skjellum, who played an active role in MPI’s design, gives a good introduction to application programming with MPI [23]. An annotated reference manual based on version 1.1 of MPI will be available by the end of 1995 [29]. A large amount of information about MPI is available via the web, including portable implementations of MPI, information about efforts to extend MPI, and publications related to MPI [8].

6 Summary

This paper has given an overview of the main features of some of the most interesting and/or widely-used approaches to programming parallel computers based on message passing. While it is not intended as a complete survey it is hoped that readers will have gained a good idea of the options open to them when using such machines. An obvious question to ask is “What is the best approach to parallel programming?” Of course, this is not a trivial question to answer, and often the depends on complicated trade-offs between performance, expressivity, ease-of-programming, maintainability, and portability. Languages such as occam and Fortran M are very good at expressing parallelism in a modular way, and offer guarantees of program correctness which may be particularly important in realtime programming. Split-C aims to give good performance through the use of bulk transfer operations while providing the convenience and ease-of-use of a global address space. Linda’s main advantages are its simplicity and expressivity, and the fact that it is a commercially-supported product. Message passing libraries are perhaps the most widely-used way of programming parallel computers. Although it is often argued that the popularity of message passing libraries rests on their flexibility and the fact that they permit the memory hierarchy to be managed to give good performance, the root cause may actually be economic and historical. When commercial parallel computers first become popular
Figure 3: One-all and all-all versions of the broadcast, scatter, and gather routines for a group of six processes. In each case, each row of boxes represents contiguous data locations in one process. Thus, in the one-all broadcast, initially just the first process contains the data $A_0$, but after the broadcast all processes contain it.
a decade ago the vendors supplied them with message passing libraries because that was (and still is) the cheapest type of parallel programming software. Users quickly developed portable message passing libraries on top of the vendor libraries, a development that ultimately led to PVM and MPI. PVM and MPI currently owe much of their popularity to the fact that they are supported on a large number of platforms, allowing software based on them to be easily ported, and they are freely available. Unfortunately no detailed study comparing different parallel programming paradigms ever seems to have been conducted. In the absence of such studies any attempt to rank them is largely subjective.

References

[8] The URL of a good MPI web page is http://www.mcs.anl.gov/mpi/. This has links to other extensive MPI pages at Mississippi State Engineering Center and Oak Ridge National Laboratory.


Fortran 90 - A thumbnail sketch

Michael Metcalf  
*CERN, Geneva, Switzerland.*

**Abstract**

The main new features of Fortran 90 are presented.

**Keywords:** Fortran

1 New features

In this brief paper, we describe in outline the main new features of Fortran 90 as compared to its predecessor, FORTRAN 77. Full details are given in [1]. Compilers for Fortran 90 are now available on all major systems, further information being accessible on the World Wide Web at the URL [http://www.fortran.com/fortran/market.html](http://www.fortran.com/fortran/market.html).

1.1 Source form

The new source form allows free form source input, without regard to columns. Comments may be in-line, following an exclamation mark (!), and lines which are to be continued bear a trailing ampersand (&). The character set is extended to include the full ASCII set, including lowercase letters. The underscore character is accepted as part of a name which may contain up to 31 characters. Thus, one may write a snippet of code such as

```fortran
SUBROUTINE CROSS_PRODUCT (x, y, z)  ! Z = X*Y  
  z1 = x(2) * y(3) - 
  x(3) * y(2)
```

Blanks are significant in the new source form, and may not be embedded inside tokens, such as keywords and variable names.

1.2 Alternative style of relational operators

An alternative set of relational operators is introduced. They are

- `<` for `.LT.`  
- `<=` for `.LE.`  
- `==` for `.EQ.`  
- `=/=` for `.NE.`

enabling us to write statements such as

```fortran
IF (x < y .AND. z**2 /= radius_squared) THEN
```

1.3 Specification of variables

The existing form of type declaration is extended to allow all the attributes of the variables concerned to be declared in a single statement. For instance, the statement

```fortran
REAL, DIMENSION(5,5), PARAMETER :: a = (/ (0., i = 1, 25) /),  &
     b = (/ (i, i = 1, 25) /)
```

declares the two objects `a` and `b` to be named constant arrays, whose values are specified by the array constructors (see Section 1.12) following the equals signs.

For those who wish to define explicitly the type of all variables, the statement

```fortran
IMPLICIT NONE
```

turns off the usual implicit typing rules.
1.4 Parameterized data types

All the intrinsic data types are now parameterized. The way this is done is best explained by examples. Suppose a given processor gives access to two kinds of integers — a default kind, represented by four bytes, and a short kind represented by two bytes. The processor might designate the default kind by a kind type value of 4, and the short kind by a value of 2. This value can be associated with the kind type parameter of a given variable as shown:

```
INTEGER(KIND=4) :: i       ! same as default
INTEGER(KIND=2) :: j       ! half the size
```

Constants corresponding to these kinds may be written. The value 99 might also be written as 99_4; a short integer constant is 63_2. It is, of course, not very portable to use explicit kind type values, and they can be parameterized, in this example for instance as

```
INTEGER, PARAMETER :: short=2
INTEGER(short)     :: j
```

A similar, optional facility is available for logical entities which might pack down into bits or bytes, depending on the value of the kind type parameter, and for character entities, in order to accommodate non-European languages whose characters require more than one byte for their representation, e.g. Kanji.

In the case of reals, at least two kinds must be provided, corresponding to the present single- and double-precision features. The kinds can be defined in terms of a range and/or precision, allowing a reasonable measure of portability. To define an array whose elements have a range of at least $10^{299}$ and a precision of at least nine decimal digits, we might write

```
INTEGER, PARAMETER :: long = SELECTED_REAL_KIND(9, 99)
REAL(long), DIMENSION :: a(1024)
a(1) = 1.23456789_long
```

where the intrinsic function SELECTED_REAL_KIND returns the kind value most closely but completely corresponding to the desired precision and range. This facility is of great importance for portable numerical software. Other intrinsic functions are available as part of this facility.

1.5 Binary, octal and hexadecimal constants

The standard allows binary, octal and hexadecimal constants to be used in DATA statements, and integers to be read and written in the corresponding formats:

```
INTEGER :: i, j
DATA j / Z’abcd’/
• WRITE (unit, ’(Z4)’) i
```

1.6 The POINTER attribute

Pointers have been added to Fortran in the form of an attribute, (rather than as a separate data type). The specification of an object with the pointer attribute designates a descriptor of the object, rather than the object itself. No storage is reserved — it has to be allocated explicitly.

A pointer may reference any dynamic object, such as an array whose actual bounds are defined at run-time, or any static object defined with the TARGET attribute. The use of the name of the pointer is, in most contexts, such as expressions or I/O lists, interpreted as a reference to the value of the target. This is known as automatic dereferencing, i.e. no explicit notation using a
pointer symbol is required. This means that existing code can readily be modified to use pointers, as only the specification statements need to be changed. Simple examples are:

```
REAL, POINTER :: arrow ! may point to a real scalar
REAL, DIMENSION(:,,:), POINTER :: pfeil ! may point to a 2D array
:
ALLOCATE(pfeil(10, 10)) ! allocate storage to pfeil
:
! define parts of pfeil
pfeil(1,1) = pfeil(5,5) + pfeil(10,10) ! the target values are used
```

Pointers are strongly typed, that is, they may point only at objects of the same type and type parameters as the ones with which they themselves are defined. Sometimes, of course, it is necessary to change the value of a pointer. For this case, a *pointer assignment* is foreseen. An example is shown in

```
REAL, DIMENSION(:,,:), POINTER :: a, b, c
:
ALLOCATE (a(10,10), b(10,10)) ! give actual space to a and b
:
! define a and b
:
c => a ! c now points to the array a
DO i = 1, 2
   IF (i == 2) c => b ! change target of c, no copy of data
   :
   :
! operations on c
:
END DO
:
```

Pointers may be components of a structure (see Section 1.15), and of the same type as the structure. In other words, a recursive definition is allowed which enables structures representing lists, trees and graphs to be defined. An example is

```
TYPE pixel
   REAL x, y
   INTEGER colour
   TYPE (pixel), POINTER :: next
END TYPE pixel
```

Pointers become associated with a target by execution of a pointer assignment or of an ALLOCATE statement. They become disassociated by, in general, execution of a NULLIFY statement. Storage occupied by a target may be returned to the processor by execution of a DEALLOCATE statement; this can result in a pointer being left ‘dangling’, and programmers must guard against their further use until they are newly associated. An intrinsic inquiry function, ASSOCIATED, indicates whether an association exists between a pointer and its target, or whether a pointer is associated at all.

Pointers may be used for static objects. In order that this should not inhibit optimization, any static object which might become a pointer target must be declared with the TARGET attribute. The compiler then knows which assumptions it can and cannot make about the references to static objects.
1.7 The CASE construct

The CASE construct allows the execution of zero or one block of code, selected from several blocks, depending on the value of an integer, logical, or character expression as compared against a set of constant values. An example is

```fortran
SELECT CASE(3*i-j) ! integer expression
    CASE(0) ! for 0 (constant value)
        ! executable code
    CASE(2,4:8) ! for 2, 4 to 8 (constant values)
        ! executable code
    CASE DEFAULT ! for all other values
        ! executable code
END SELECT
```

The default clause is optional; overlapping ranges are not permitted.

1.8 The loop construct

A new loop construct is introduced. In a simplified form, its syntax is

```fortran
[name:] DO [(control)]
    block of statements
END DO [name]
```

(where square brackets indicate optional items). If the control parameter is omitted, an endless loop is implied; if present, it has the form \(i = \text{intexp1}, \text{intexp2} [, \text{intexp3}]\). The optional name may be used in conjunction with CYCLE and EXIT statements to specify which loop in a set of nested loops is to begin a new iteration or which is to be terminated, respectively.

The DO...WHILE has also been incorporated into the language. It is redundant, as the DO construct and the EXIT statement provide the same functionality.

1.9 Program units

An enhanced form of the call to a procedure allows keyword and optional arguments, with intent attributes. For instance, a subroutine beginning

```fortran
SUBROUTINE solve (a, b, n)
    OPTIONAL, INTENT(IN) :: b
```

might be called as

```fortran
CALL solve (n = i, a = x)
```

where two arguments are specified in keyword (rather than positional) form, and the third, which may not be redefined within the scope of the subroutine, is not given in this call. The mechanism underlying this form of call requires an interface block containing the relevant argument information to be specified.

Procedures may be specified to be recursive, as in

```fortran
RECURSIVE FUNCTION factorial(x)
```

The old form of the statement function is generalized to an internal procedure which allows more than one statement of code, permits variables to be shared with the host procedure, and contains a mechanism for extending operators and assignment for use with derived-data types. This we shall return to in Section 1.16.
1.10 Extensions to CHARACTER data type

A number of extensions to the existing CHARACTER data type, in addition to the optional
kinds, permit the use of strings of zero length and the assignment of overlapping substrings, and
introduce new intrinsic functions, such as TRIM, to remove the trailing blanks in a string. Some
intrinsics, including INDEX, may operate, optionally, in the reverse sense.

1.11 Input/Output

The work in the area of input/output has been mainly on extensions to support the new data types,
and an increase in the number of attributes which an OPEN statement may specify, for instance
to position a file or to specify the actions allowed on it.

In addition, there are new specifiers on READ and WRITE statements. One is the ADVANCE specifier for formatted, sequential files, allows data to be read and written without being
concerned with any record structure. It is, for instance, possible to read a specified number of
characters from part of a record, to remain positioned at that point in the record, and to read from
the remainder with a subsequent read. At present, a single read would cause the whole record to
be read, regardless of the length of the input list.

1.12 Array processing

The array processing features are one of the most important new aspects of the language.

An array is defined to have a shape given by its number of dimensions, or rank, and the
extent of each dimension. Two arrays are conformable if they have the same shape. The operat-
ions, assignments and intrinsic functions are extended to apply to whole arrays on an element-
by-element basis, provided that when more than one array is involved they are all conformable.
When one of the variables involved is a scalar rather than an array, its value is distributed as
necessary. Thus we may write

```
REAL, DIMENSION(5, 20) :: x, y
REAL, DIMENSION(-2:2, 20) :: z

z = 4.0 * y * SQRT(x)
```

In this example we may wish to include a protection against an attempt to extract a negative
square root. This facility is provided by the WHERE construct:

```
WHERE (x >= 0.)
  z = 4.0 * y * SQRT(x)
ELSEWHERE
  z = 0.
END WHERE
```

which tests x on an element-by-element basis.

A means is provided to define array sections. Such sections are themselves array-valued
objects, and may thus be used wherever an array may be used, in particular as an actual argument
in a procedure call. Array sections are selected using a triplet notation. Given an array

```
REAL, DIMENSION(-4:0, 7) :: a
```

the expression a(-3,:) selects the whole of the second row (i.e. the elements a(-3,1),... a(-3,7)), and a(0:-4: -2, 1:7:2) selects in reverse order every second element of every second
column.

Just as variables may be array-valued, so may constants. It is possible to define a rank-one
array-valued constant as in

```
5
```
and to reshape it to any desired form:

```fortran
REAL, DIMENSION(2, 3) :: a
a = RESHAPE(/1, I, I = 1, 6, /), (/2, 3/)
```

where the second argument to the intrinsic function defines the shape of the result, and the first
defines an array of the first six natural numbers, using the implied-DO loop notation familiar in
I/O lists.

Many vector processors have a gather/scatter capability. This can be directly accessed in
Fortran 90 by use of the vector-valued subscript notation:

```fortran
INTEGER, DIMENSION(1024) :: i, j
REAL, DIMENSION(1024) :: a, b
```

```fortran
! Define i, j and a
b(j) = a(i) ! permute the elements of a into b
```

### 1.13 Dynamic storage

Fortran 90 provides three separate mechanisms for accessing storage dynamically. The first mech-
anism is via the ALLOCATE and DEALLOCATE statements which, as their names imply, can
be used to obtain and return the actual storage required for an array whose type, rank, name, and
allocatable attribute have been declared previously:

```fortran
REAL, DIMENSION(:) :: x
ALLOCATE(x(n:m)) ! n and m are integer expressio ns
x(j) = q
CALL sub(x)
```

```fortran
DEALLOCATE (x)
```

Unless an allocated array has the SAVE attribute, it becomes undefined whenever a RETURN
or END statement is executed in the procedure in which it is declared. For good housekeeping,
such arrays should be explicitly deallocated. The fact that allocation and deallocation can occur
in random order implies an underlying heap storage mechanism.

The second mechanism, for local arrays with variable dimensions, is the automatic array:

```fortran
SUBROUTINE sub(i, j, k)
REAL, DIMENSION(i, j, k) :: x ! bounds from dummy arguments
whose actual storage space is provided (on a stack) when the procedure is called.
```

Finally, we have the assumed-shape array, whose storage is defined in a calling procedure
(so it is not strictly dynamic), and for which only a type, rank, and name are supplied:

```fortran
SUBROUTINE sub(a)
REAL, DIMENSION(:,:) :: a
```

Various enquiry functions may be used to determine the actual bounds of the array:

```fortran
DO i = LBOUND(a, 3), UBOUND(a, 3)
  DO j = LBOUND(a, 2), UBOUND(a, 2)
    DO k = LBOUND(a, 1), UBOUND(a, 1)
where LBOUND and UBOUND give the lower and upper bounds of a specified dimension, re-
spectively.
```
1.14 Intrinsic procedures

Fortran 90 defines about 100 intrinsic procedures. Many of these are intended for use in conjunction with arrays for the purposes of reduction (e.g. SUM), inquiry (e.g. RANK), construction (e.g. SPREAD), manipulation (e.g. TRANSPOSE), and location (e.g. MAXLOC). Others allow the attributes of the working environment to be determined (e.g. the smallest and largest positive real and integer values), and access to the system and real-time clocks is provided. A random number subroutine provides a portable interface to a machine-dependent sequence, and a transfer function allows the contents of a defined area of physical storage to be transferred to another area without type conversion occurring.

Of particular note is that the bit string manipulation functions and subroutine of MIL-STD 1753 have been incorporated into the language; their definitions have been extended to handle integer arrays in a completely upwards-compatible fashion.

1.15 Derived-data types

Fortran has hitherto lacked the possibility of building user-defined data types. This will be possible in Fortran 90, using a syntax illustrated by the example

```
TYPE staff_member
  CHARACTER(LEN=20) :: first_name, last_name
  INTEGER :: id, department
END TYPE
```

which defines a structure which may be used to describe an employee in a company. An aggregate can be defined as

```
TYPE(staff_member), DIMENSION(1000) :: staff
```

defining 1000 such structures to represent the whole staff. Individual staff members may be referenced as, for example, `staff(no)`, and a given field of a structure as `staff(no)%first_name`, for the first name of a particular staff member. More elaborate data types may be constructed using the ability to nest definitions, for instance, to build a structure to define companies:

```
TYPE company
  CHARACTER(LEN=20) :: name
  TYPE(staff_member), DIMENSION(1000) :: staff
END TYPE

TYPE(company), DIMENSION(20) :: companies
```

1.16 Data abstraction

It is possible to define a derived-data type, and operations on that data type may be defined in an internal procedure. These two features may be combined into a module which can be propagated through a whole program to provide a new level of data abstraction (see also [2]). As an example we may take an extension to Fortran’s intrinsic CHARACTER data type whose definition is of a fixed and pre-determined length. This user-defined derived-data type defines a set of modules to provide the functionality of a variable length character type, which we shall call string. The module for the type definition might be

```
MODULE string_type
  TYPE string(maxlen)
    INTEGER :: length
    CHARACTER(LEN=maxlen) :: string_data
  END TYPE String
```

7
END MODULE String_type
With

USE string_type
TYPE(string(60)), DIMENSION(10) :: cord
we define an array of 10 elements of maximum length 60. An actual element can be set by
cord(3) = 'ABCD'
but this implies a redefinition, or extension, of the assignment operator to define correctly both fields of the element. This can be achieved by the internal procedure

SUBROUTINE c_to_s_assign(s,c)
  TYPE (string) :: s
  CHARACTER(LEN=*) :: c
  s%string_data = c
  s%length = LEN(c)
END SUBROUTINE c_to_s_assign

and the interface block

INTERFACE ASSIGNMENT(=) ! Extend assignment operator
  MODULE PROCEDURE c_to_s_assign
END INTERFACE

which can both be included in the module, together with other valid functions such as concatenation, length extraction, etc., to allow the user-defined string type to be imported into any program unit where it may be required, in a uniformly consistent fashion.

2 Backwards compatibility

The procedures under which standards are developed require that a period of notice be given before any existing feature is removed from the language. This means, in practice, a minimum period of one revision cycle, which for Fortran means a decade or more. The need to remove features is evident: if the only action of the standards committee is to add new features, the language will become grotesquely large, with many overlapping and redundant items. The solution adopted was to publish as an Appendix to the standard a set of two lists showing which items have been removed or are candidates for eventual removal.

The first list contains the Deleted Features (those which in the previous standard were listed as Obsolescent Features) which have now been removed. There are no Deleted Features in Fortran 90 which contains the whole of FORTRAN 77.

The second list contains the Obsolescent Features, those considered to be redundant and little used, and which should be removed in the next revision (although that is not binding on a future committee). The Obsolescent Features are:

Arithmetic IF       Real and double precision DO variables
Shared DO termination DO termination not on CONTINUE (or ENDDO)
Alternate RETURN    Branch to ENDIF from outside block
PAUSE                ASSIGN and assigned GOTO

References

High Performance Fortran

Michael Metcalf
CERN, Geneva, Switzerland.

Abstract

This paper presents the main features of High Performance Fortran (HPF), a means to program data-parallel programs in a machine-independent way.

Keywords: Fortran, Parallel processing

1 HPFF

A basic problem in programming for parallel architectures is that each machine has its own design, and also its own specific software for accessing its hardware features. In 1992, as a response to this situation, the High Performance Fortran Forum (HPFF) was founded, under the guidance of Professor Ken Kennedy, to produce a portable Fortran-based interface to parallel machines for the solution of data-parallel problems, mainly involving regular grids. Over 40 organizations participated, and the work used existing dialects, such as Fortran D (Rice University), Vienna Fortran and CM Fortran (Thinking Machines), as inspiration.

It was realised early on that much of the desired functionality already existed in the, then, very new Fortran 90 (see [1]), and this was selected as the base language on which to build HPF itself. The array processing features of Fortran 90 are particularly relevant. This enabled the goal of producing an ad hoc standard for HPF within a year to be met, and full details can be found in [2]. The standard document itself is obtainable by anonymous ftp at titan.cs.rice.edu in the directory public/HPFF/draft as the file hpf-v10-final.ps.Z. At the time of writing, HPFF has reconvened to produce a second, more advanced, version ± HPF II. One new feature is expected to be an extension for irregular grids.

The basic approach adopted was that of designing a set of directives that may be added to Fortran 90 programs, together with a few syntactical additions and some extra libraries, thus creating a data parallel programming language that is independent of the details of the architecture of the parallel computer it is run on. The principle is to arrange for locality of reference of the data by aligning related data sets to one another, and distributing the aligned sets over memory regions such that, usually, calculations on a given processor are performed between operands already on that processor. Any message passing that might nevertheless be necessary to communicate data between processors is handled by the compiler and run-time system.

2 Directives

The directives all have the form

!HPF$ directive

and are interpreted as comment lines by non-HPF processors.

2.1 Alignment

There are various, sometimes quite complicated, ways of aligning data sets. A simple case is when we want to align three conformable arrays with a fourth:

!HPF$ ALIGN WITH b :: a1, a2, a3
thus ensuring their subsequent distribution will be identical.

Although the ranks of the alignees must be the same, it is possible, using the ‘*’ notation, to collapse a dimension so enabling the extents to differ:

```hpf
REAL a(n28), b(n28), c(n28), q(n29)
!HPF$ ALIGN (*, :), WITH q :: a, b, c
```

where the ‘:*’ is a position holder for that dimension (taking elements in order). For the first dimension, the ‘*’ causes the 3, 4 or 43 elements, respectively, to be aligned with q.

For single alignees, a statement form exists. This permits, additionally, a transpose via dummy variables (here j and k):

```hpf
!HPF$ ALIGN x(j, k) WITH d2(k, j)
```

as well as, in the following example, a lower bound to be fixed (first dimension of d), a dimension to be shifted (third dimension of d), or a stride to be defined (fourth dimension of d):

```hpf
!HPF$ ALIGN a(:, *, :, :, :) WITH d(31:, :, k+3, 2:8:2)
```

### 2.2 Distribution

Having aligned the data sets with one another, the next step is to map these data objects onto a set of abstract processors. Given

```hpf
REAL salami(10000)
!HPF$ DISTRIBUTE salami(BLOCK)
```

we would, on a set of 50 abstract processors, map 200 contiguous elements to each one. This can be made more specific:

```hpf
!HPF$ DISTRIBUTE salami(BLOCK(256))
```

specifies the exact number per processor. The CYCLIC keyword is also available, to cycle the elements over the processors in turn.

For a multi-dimensional array, the methods may be combined:

```hpf
!HPF$ DISTRIBUTE three(BLOCK(64), CYCLE(128), *)
```

where, as before, the ‘*’ collapses a complete dimension.

### 2.3 Processor layout

The layout of the abstract processors may be specified as a regular grid:

```hpf
!HPF$ PROCESSOR rubik(3, 3, 3)
```

and then distributions mapped onto it:

```hpf
!HPF$ DISTRIBUTE ONTO rubik :: a, b, c
```

Using a notation we have already seen, this may be further specified, as in this statement form

```hpf
!HPF$ DISTRIBUTE a(BLOCK, CYCLIC, BLOCK(3:19:4), *)  &
```

```hpf
!HPF$  ONTO rubik ! a is rank-4
```

For a high level of portability and efficiency, it is clearly necessary to be able to enquire about the actual processor layout. For this, two new intrinsic functions provide the number of processors and the actual shape of their layout. Thus, the abstract layout may be specified in terms of the actual number available:

```hpf
!HPF$ PROCESSORS r(NUMBER_OF_PROCESSORS() / 8, 8)
```

and an array, here ps, may be defined to hold the shape of the layout, each element of ps containing the number of processors in the corresponding dimension of the layout:

```hpf
INTEGER, DIMENSION SIZE(PROCESSORS_SHAPE()) :: ps
ps = PROCESSORS_SHAPE()
```
2.4 Templates

Usually, we align arrays to one another in such a fashion that at least one of them covers the entire index space of all of them, as in

!HPF$ ALIGN a WITH b

Where it is required to make arrays partially overlap in some fashion, it would be possible to use an artificial array to support the mapping. However, after much debate, HPFF decided to incorporate this facility into the HPF language using the TEMPLATE directive. Its use is shown in

!HPF$ TEMPLATE, DISTRIBUTE(BLOCK, BLOCK) :: earth(n+1, n+1)
REAL, DIMENSION(n, n) :: nw, ne, sw, se
!HPF$ ALIGN nw(i, j) WITH earth(i, j)
!HPF$ ALIGN ne(i, j) WITH earth(i, j+1)
!HPF$ ALIGN sw(i, j) WITH earth(i+1, j)
!HPF$ ALIGN se(i, j) WITH earth(i+1, j+1)

where each of the four alignees, nw, ne, sw and se, is mapped to a different corner of the template, earth.

2.5 Dynamic alignment and distribution

The directives described so far have all had effect at compile time. By contrast, the DYNAMIC attribute:

!HPF$ DYNAMIC a, b, c, d

or

!HPF$ DYNAMIC, ALIGN WITH s :: x, y, z

allows the use, at run time, of the REALIGN and REDISTRIBUTE statements. These are similar to the corresponding directive forms, but the rules of Fortran 90 allow a more general form of subscript expressions.

3 Parallel constructs

The parallel constructs are mostly extensions to the Fortran 90 syntax. The Fortran standardization committees are likely to add these to Fortran 95, a minor revision of Fortran 90 now in preparation.

3.1 FORALL statement and construct

The FORALL is an addition to the Fortran 90 syntax that assures a compiler that the individual assignments in a statement are independent, and can therefore proceed in parallel. It also overcomes some restrictions found in ordinary array assignments, in particular that the left-hand and right-hand sides of assignments must be conformable arrays. Examples of the statement form are:

FORALL(i = 1:n, j = 1:m) a(i, j) = i + j
FORALL(i = 1:n) a(i, i) = x(i)
FORALL(i = 1:n, j = 1:n, y(i, j) /= 0.) x(j, i) = 1.0/y(i, j)

The construct form allows, in addition, a sequence of independent statements to be executed in order and once only. In

FORALL(i = 2:n-1, j = 2:n-1)
    a(i, j) = a(i, j-1) + a(i, j+1) + a(i-1, j) + a(i+1, j)
the second assignment will not begin until the first has completed for all values of i and j, and will then use the newly computed values.

### 3.2 PURE attribute

An obstacle to generating parallel code in the presence of function references is that non-intrinsic functions may have side effects that potentially change the results of subsequent assignments. Within a FORALL statement or construct, the programmer is able to make a pact with the compiler, asserting that the function referenced has no side effects and may be safely referenced in parallel invocations. This is achieved by giving such functions the PURE attribute, a further Fortran 90 syntax extension. Given

```fortran
PURE FUNCTION my_func(i)
  ! We can invoke
  FORALL(i = 1:n) a(i) = my_func(i)
```

We are saying that my_func does nothing other than return a result, and in particular that it does not change the value of its argument, performs no I/O, and modifies no global variable (e.g. in a module).

### 3.3 Parallel loops

Unless it can determine otherwise by dependency analysis, a compiler has to make the assumption that the individual statements of a DO or FORALL construct depend on one another. It is possible in HPF to insert a directive that asserts that each iteration or statement is, in fact, independent of all others, as in

```fortran
!HPF$ INDEPENDENT
DO i = 1, n
  a(p(i)) = b(i)  ! p is a permutation
END DO
```

where, as p is a permutation, all assignments are independent and can proceed in parallel. In nested loops, each one requires its own directive, where appropriate.

### 4 HPF intrinsic and library procedure

The Fortran 90 intrinsic functions are augmented by a further three for use in a parallel environment, and by an HPF Library of procedures. Their large number means they cannot be described here, and the interested reader is referred to the standard or to [2]. Suffice it to list their principal groupings: to determine array mappings, additional bit manipulation functions, additional array reduction functions, array sorting, array scatter functions, and two sets of partial array reduction functions.

### 5 Extrinsic procedures

HPF introduces the notion of extrinsic procedures. This defines both an interface to non-HPF procedures, or even languages, and a mechanism for implementing the SPMD programming model. For this latter purpose it is possible to pass parts of a decomposed array to local procedures on each processor, and the extrinsic procedure thus defined terminates when each local
copy has finished executing on its own part of the array. Any communications require explicit management.

6 Storage association

FORTRAN 77 is based on storage association through the use of, in particular, COMMON and EQUIVALENCE. Use of these facilities makes parallelization of programs difficult, as otherwise unrelated variables may appear to be related because they are, for instance, in the same COMMON block. Fortran 90 provides new facilities for avoiding storage association, and so it should not be used in new programs. However, for legacy programs, the NOSEQUENCE directive is provided, asserting that certain apparent dependencies do not, in fact, exist.

7 Subset HPF

In order to enable useful implementations of HPF to be produced quickly, a subset language was defined. This consists of Fortran 90 apart from modules, derived data types, the CASE construct, etc., plus the HPF extensions without dynamic realignment and distribution, certain complicated alignments, the PURE attribute, the FORALL construct, the HPF Library, and extrinsic procedures. Early users are advised, initially, to stick to this subset language.

8 Conclusion

This paper has briefly described the main features of HPF, in particular those for aligning and distributing regular grids of data over the processors of a parallel computer. HPF-conformant compilers are now available, for instance from DEC and IBM, and other vendors offer HPF preprocessors. However, as far as high-energy physics is concerned, we note that only within the area of accelerator design do there appear to be suitable applications for using HPF. Application programs for processing experimental data do not contain the large regular grid patterns that HPF is designed to handle. We can thus expect very little use of HPF in our field.

References

Using a Statistics Package

Roger Barlow  
Department of Physics and Astronomy, Manchester University, Manchester, UK

Abstract

The facilities provided by a statistics package are described, using Genstat [1] as an example, with some students’ exam results as illustrative material.

Keywords: Statistics; Statistics Packages; Genstat; Two sample problem; ANOVA; Regression

1 Serious Statistics

Physicists don’t need serious statistics. The laws of physics are fundamentally simple: the current through a resistance depends only on the applied voltage, the cross section for $e^+e^-$ annihilation depends only on the cms energy. It’s the ‘soft’ sciences where the laws are complicated – crop yield depends on the amount of fertiliser and irrigation and sunshine and seed type and pests present... many variables being uncontrollable, perhaps unmeasurable, perhaps undefinable – that drove the development of statistics in the past, and need to use it seriously today.

Statistics packages do serious statistics. So if you have a well-defined physics problem, for example a set of points to which you want to fit a straight line, or a parabola, or even a set of quartic splines, don’t try to use a statistics package: your best route is to write your own program, calling on the resources of a good subroutine library [2, 3], and perhaps a textbook that explains elementary statistics in simple language that physicists can relate to [4, 5, 6, 7].

But even though Statistics Packages aren’t useful for physics, they can still (like word-processors and e-mail and spreadsheets) be useful for physicists. If you have a set of measurements you want to explore to look for trends and connections, not really knowing what the right questions are to ask, then a Statistics Package is the appropriate tool.

Several Statistics Packages are on the market. I’ve chosen Genstat$^1$ [1, 8] as an example. There are reasons for doing so: it came out well in a survey run by the journal Physics World [9], it is supported by the well-known Numerical Algorithms Group, and it comes from the Rothamsted Experimental Agricultural Station, which played a major part in the development of serious statistics. But the aim of these lectures is not to teach you about a particular product, but to show you the sorts of things such packages can do, so that when you encounter a suitable set of data you will recognise how it can be analysed, perhaps with Genstat or perhaps some other statistics package which will have similar facilities even if the language to use them is different$^2$.

As an example of random data within which there are trends and patterns of an unknown nature I have taken the performance of some first year physics students. Some students do well, some do badly. It’s very important for us as teachers to understand why this is, and to identify causes of problems. A statistical analysis of the results can help.

---

$^1$Genstat is a trademark of the Lawes Agricultural Trust

$^2$For this reason, technical details of the language syntax are omitted from these proceedings, though they were given in the lectures to enable the summer school participants to use Genstat in a practical session.
The total assessment marks (expressed as percentages) for some first year students are shown as a histogram in figure 1. It’s relevant to point out that marks above 70% are reckoned excellent, whereas marks below 40% are reckoned to be failures.

The set of marks was entered into a vector \(^3\) of numerical values, called in Genstat a variate, named mark, and then plotted using DHIST mark. Binning is chosen automatically: it can be over-ridden if desired.

One histogram doesn’t tell us very much: let’s look further. These marks were actually obtained from two classes of students, one doing a degree in Physics and the other in Astrophysics; the two courses are very similar in their first year. We can ask whether the performance of the two classes is the same.

You might think that the obvious way to model the data was to have two separate objects: mark_A and mark_B. But in fact it’s better to have one object mark that contains all the marks with a second object class that specifies the class to which this student belongs. These could be (and are) represented as columns on a datafile

\[
\begin{array}{ll}
A & 75.2 \\
A & 45.2 \\
B & 60.4 \\
A & 37.3 \\
B & 45.6 \\
\ldots
\end{array}
\]

Why is this better? First because everything in the mark object is of the same nature - in this case, percentages, which means they have numerical values between 0 and 100. If you use mark_A and mark_B it is not implicit in the data representation that they contain quantities of the same nature. Secondly because if more classes are introduced this does not need more objects,

\(^3\)Genstat calls such objects vectors, which is not really accurate as they do not satisfy the vector space axioms, but this is a quibble.

---

**Figure 1**: Distribution of Students’ marks

2 Data

The total assessment marks (expressed as percentages) for some first year students are shown as a histogram in figure 1. It’s relevant to point out that marks above 70% are reckoned excellent, whereas marks below 40% are reckoned to be failures.

The set of marks was entered into a vector \(^3\) of numerical values, called in Genstat a variate, named mark, and then plotted using DHIST mark. Binning is chosen automatically: it can be over-ridden if desired.

One histogram doesn’t tell us very much: let’s look further. These marks were actually obtained from two classes of students, one doing a degree in Physics and the other in Astrophysics; the two courses are very similar in their first year. We can ask whether the performance of the two classes is the same.

You might think that the obvious way to model the data was to have two separate objects: mark_A and mark_B. But in fact it’s better to have one object mark that contains all the marks with a second object class that specifies the class to which this student belongs. These could be (and are) represented as columns on a datafile

\[
\begin{array}{ll}
A & 75.2 \\
A & 45.2 \\
B & 60.4 \\
A & 37.3 \\
B & 45.6 \\
\ldots
\end{array}
\]

Why is this better? First because everything in the mark object is of the same nature - in this case, percentages, which means they have numerical values between 0 and 100. If you use mark_A and mark_B it is not implicit in the data representation that they contain quantities of the same nature. Secondly because if more classes are introduced this does not need more objects,
just further possible values (C, D, E...) in class.

The data in the vector class are very different from those in mark: each specifies the category (or group or class) of the measurement. They are clearly not numeric - even though we could have used 1, 2, 3... rather than A, B, C... to label the categories. They are also not textual: the actual values of the labels are irrelevant. A vector of categories is called a factor, and these are very heavily used.

Your statistical analysis is generally assumed to work on columns of data - numerical or categorical - like these. Your complete set of measurements constitutes a list of vectors, all of the same length. Most Genstat commands operate readily on single vectors or on lists of vectors, separated by commas: you can say PRINT mark or PRINT mark, class.

The point to be watched is that horizontal linkage is not hard-wired in the language. There is nothing except your own common sense to stop you saying (because you’d like the marks in order) SORT mark. That makes mark and class out of step. SORT mark, class has the correct effect, as SORT sorts all vectors in its list into the order specified by the first one.

3 The Two Sample Problem

The distributions of marks from the two classes appear to be different. This can be seen in figure 2 which shows box-and-whisker plots (generated by BOXPLOT mark; class). These show the range of the measurements as a line, the box covers the interquartile range (i.e. it contains the central 50% of the measurements) and the horizontal bar is the median.

![Figure 2: Marks for Physics and Astrophysics classes](image)

3.1 The *t* test

To quantify this disagreement (i.e. to see if it’s genuine or a fluctuation) one can use *Student’s t test*. The command TTEST [GROUP=class] mark gives the result

```
***** Two-sample T-test *****
       Sample  Size  Mean  Variance
       Y_1    86    55.58   190.8
       Y_2    31    47.08   236.3
```
*** Test for evidence that the distribution means are different ***

Test statistic $t=2.85$ on 115 df.

Probability level (under null hypothesis) $p=0.005$

The arithmetic means are evaluated and differ by 8.5. The spread of the results provides an estimate of the error on this difference and we are told that 8.5 is 2.85 times this estimated error: this ratio is called $t$. The probability of such a large discrepancy happening by chance is clearly small; it’s not quite the same as a $2.85\sigma$ Gaussian significance because this is only the estimated error, not the true $\sigma$. Instead the significance is given by the Student’s $t$ significance for 115 degrees of freedom, which is 0.5% - so it really looks as if there is a meaningful difference..

3.2 The Mann-Whitney Test

Strictly speaking, Student’s $t$ assumes that the distributions concerned are Gaussian (Normal). Clearly – looking at Figure 1 – this is not true. Actually, Student’s $t$ is surprisingly robust, and works pretty well even for moderately non-Gaussian distributions, so the significance of the difference probably isn’t invalidated. But an alternative is at hand in the Mann-Whitney test, (also known as the U test and as Wilcoxon’s rank sum test) which also tests for the difference of the means of two distributions, but makes no assumptions about their shape. The syntax for this is similar: MANNWHITNEY [GROUPS=class] mark and the results appear as

Mann-Whitney U (Wilcoxon Rank-Sum) test

Value of U = 901.0 (first variate has highest rank score)

Normal Approximation = 2.668 (p=0.01)

Adjusted for ties = 2.668 (p=0.01)

Sample sizes: 86 31

The two samples are ranked together in order, and $U$ is the number of times an A item comes after a B (counting all the $N_A N_B$ pairs). If the distributions were the same then on average $U$ would be $N_A N_B / 2$, which is 1333, but it’s only 901. The difference of 432 is significant at the 2.67 standard deviation level, and the probability of that happening by chance is only 1%. Actually this is an approximation, but it’s valid for sample sizes above 5. If any of the scores are equal (tied) then various adjustments have to be made, and are given.

3.3 The Kolmogorov-Smirnov test

The Kolmogorov-Smirnov 2 sample test is sensitive to differences in distributions even if they have the same mean. The syntax is similar, KOLMOG2 [GROUPS=class] mark

Kolmogorov-Smirnov Two-Sample Test

Maximum Difference = 0.3623

Chi-squared (2 df) = 11.97 (p=0.00)

Sample sizes: 86 31

In this test the cumulative distributions are formed for both samples, and normalised to 1. Then the largest difference – here a whopping 0.3623 – is found between the two. This can be translated into a $\chi^2$ value of 12 for 2 degrees of freedom – and the probability of that happening at random is negligible. This result supports the others. We are forced to conclude that Physics students perform better than those studying Astrophysics.
4 Analysis of variance

To continue exploring, we can look at results from several classes (Physics with Theoretical Physics, Physics with Business Studies, and others.) The box-and-whisker plots (figure 3) don’t tell us too much.

The $t$ test only works for the two sample problem. For many samples one uses the Analysis of variance. This proceeds as follows. (Serious statisticians use ‘Variance’, the squared deviation from the mean, rather than the r.m.s. deviation more familiar to physicists.)

Suppose the $N$ measurements have a mean $m$ (the global mean). They are divided into $N_S$ samples, with sample $s$ containing $n_s$ measurements and having a sample mean $m_s$. The total squared deviation from the mean can be split into two parts:

$$\sum_{i=1}^{N}(x_i - m)^2 = \sum_{s=1}^{N_S} \sum_{i \in s} (x_i - m)^2$$
$$= \sum_{s} \sum_{i \in s} ((x_i - m_s) + (m_s - m))^2$$
$$= \sum_{s} \sum_{i \in s} (x_i - m_s)^2 + \sum_{s} n_s(m_s - m)^2 + 2 \sum_{s} (m_s - m) \sum_{i \in s} (x_i - m_s)$$

the first term is the within-sample variance, the second the between-sample variance, and the third vanishes.

Now suppose that there is really no difference between the samples. All the measurements are drawn from a Gaussian distribution with some mean and some standard deviation $\sigma$. The ratio of the between-sample variance and the within-sample variance (with the numbers of degrees of freedom included to keep things tidy) is

$$\frac{\sum_{s} n_s(m_s - m)^2}{\sum_{i=1}^{N} (x_i - m_s)^2} = \frac{\sum_{s} \frac{(m_s - m)^2}{\sigma^2/n_s}}{\sum_{i=1}^{N} \frac{(x_i - m_s)^2}{\sigma^2}}$$

Notice the cunning way the distributions can be divided by $\sigma^2$ even though we don’t know what it is! The top and bottom are both a $\chi^2$ per degree of freedom, so (under the null hypothesis)
they should both be 1 on average, and their ratio should also be 1. If the ratio is significantly bigger than 1, that implies that the variation between samples is bigger than expected from the variation within samples, and that they are really different. The significance is calculated using Snedecor’s $F$ test.

Analysis of Variance (ANOVA) takes two commands. First you specify the samples by TREATMENTS class – Genstat is betraying its agricultural origins here; ANOVA is much used in comparing crop yields from different plots of land treated with different fertilisers, pesticides, etc. Then you do the analysis by ANOVA [FPROB=yes] mark. The results are:

<table>
<thead>
<tr>
<th>Source of variation</th>
<th>d.f.</th>
<th>s.s.</th>
<th>m.s.</th>
<th>v.r.</th>
<th>F</th>
<th>pr.</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>7</td>
<td>4590.4</td>
<td>655.8</td>
<td>2.88</td>
<td>0.007</td>
<td></td>
</tr>
<tr>
<td>residual</td>
<td>167</td>
<td>38015.3</td>
<td>227.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>174</td>
<td>42605.6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

This shows that the total sum of squares of deviations is 42605.6. The within-sample part is 38015.3, which on dividing by the 167 degrees of freedom says that the variance of a measurement is 227.6, i.e. $\sigma = \sqrt{227.6} = 15.1$. The between-sample part is 4590.4 (note that 4590.4+38015.3=42605.6) and as there are 8-1=7 degrees of freedom for this, the estimate of the variance you get from the group means is 4590.4/7=655.8. This is bigger than 227.6 by a factor 2.88. The probability of this happening by chance, as calculated from Snedecor’s $F$ distribution, is a mere 0.7%, giving strong support to the theory that there must be something else at work that leads to variation between the samples.

5 Some manipulation

Suppose we decide we know class B is different anyway so let’s leave that out. This can be done using the RESTRICT command:

```
RESTRICT mark;class.IN.!T(A,C,D,E,F,G,H)
TREATMENTS class
ANOVA mark
```

The values in the variate mark are restricted to those for which the corresponding member of class are in the list of text characters. This is temporary: the command RESTRICT mark will switch all members back on again.

Suppose we wanted to compare class C against all other classes combined using the $t$ test (which is only applicable to 2 samples). A logical expression actually gives a numerical value: 1.0 for true and 0.0 for false. So we can compare class C against the rest by:

```
FACTOR [LEVELS=2] newclass
CALCULATE newclass=1+class.IN.!T(C)
TTEST [GROUP=newclass] mark
```

newclass must be declared as a factor (with 2 levels) before use, otherwise CALCULATE will create it as a variate of numerical data.

---

4 Yes, that’s the opposite way round from the C language convention
6 Correlation

The data also contain, in the variate `alevel`, the students’ performance in the exams they took at the end of their school career, before entry to university. These are coded as a number with a maximum value of 30. Some students did not come through the conventional (English) system, and no such figure is available. These are coded as ‘missing values’, using asterisks. Genstat can handle these.

One would expect some link between present and past performance. This can be shown using `DGRAPH mark;alevel;` which produces figure 4.

![Exam result versus A level score](image)

**Figure 4:** Exam result versus A level score

This confirms that there is a connection, but it is (perhaps?) not as strong as one might expect.

During the year the students do practical laboratory work, and this mark is available in `lab`. They also attend – or are supposed to attend – two tutorials every week: the actual number attended is available in `attend`. These can also be plotted. For a numerical result we can look at the correlation between the various values. However, before we do that, remember that the lab work is included in the final mark, which will bias the correlation. To remove this we have to know how it was included, and the details vary. For classes A and B the lab counts for 200 out of a final mark of 875, so we say (remembering that these are all percentages)

```genstat
RESTRICT mark;class.IN.!T(A,B)
CALCULATE newmark=(mark*875-lab*200)/675
CORRELATE [print=correlate] newmark,attend,alevel,lab
```

This gives the result

<table>
<thead>
<tr>
<th></th>
<th>newmark</th>
<th>attend</th>
<th>alevel</th>
<th>lab</th>
</tr>
</thead>
<tbody>
<tr>
<td>newmark</td>
<td>1.000</td>
<td>0.605</td>
<td>0.561</td>
<td>0.660</td>
</tr>
<tr>
<td>attend</td>
<td>0.605</td>
<td>1.000</td>
<td>0.180</td>
<td>0.724</td>
</tr>
<tr>
<td>alevel</td>
<td>0.561</td>
<td>0.180</td>
<td>1.000</td>
<td>0.262</td>
</tr>
<tr>
<td>lab</td>
<td>0.660</td>
<td>0.724</td>
<td>0.262</td>
<td>1.000</td>
</tr>
</tbody>
</table>

newmark attend alevel lab
notice the surprisingly low correlation between the Alevel performance and the lab mark and attendance, and the high correlation between attendance, lab, and the rest of the marks. This suggests that the lab mark and the attendance are both measures of how conscientiously a student works, and that the first year exam marks depend on this very strongly, indeed that this is more important than previous performance.

7 Regression

Data like that in figure 4 cries out to have a straight line draw through it. In serious statistics this is known as regression [10, 11].

7.1 Historical interlude

The reason for this name – regress is the opposite of progress – is historical but still quite significant. In the last century Galton measured the heights of fathers and their (adult) sons, plotted one against the other and drew a least squares straight line through them. He found that there was a clear trend for tall fathers to have sons who were taller than average - but not as tall as their fathers. Likewise the children of short fathers tended to be shorter than the average, but closer to the average than their fathers. He saw this – rather gloomily – as a regressive trend towards mediocrity.

This well-worn example does illustrate three points about ‘regression’ and why it’s not just the same as the straight line fitting you’re used to (even though the formulae are the same).

- The errors on the measurements are irrelevant. If Galton had used a laser-interferometer to measure the heights to a fraction of a micron, the results would not change.
- The basic law – if there is one – is not \( h_s = F(h_f) \) but \( h_s = F(h_f, h_m, f, e...) \). The son’s height does not just depend on the father’s height but on many other variables (mother’s height, feeding, exercise...). The spread in all the other variables is what gives rise to the spread in \( h_s \) for a fixed \( h_f \).
- There is a clear causal link in the data. It is also true that tall sons tend to have tall fathers, but they don’t cause them! In regression there is generally one parameter which is regarded as being caused – the response variable – plotted as the vertical axis, and another, or others, called the control variables which are assumed to influence the response. Sometimes, like the amount of fertiliser, they can be controlled, sometimes, like the amount of rain, they can’t.

7.2 Doing simple regression

Regression, like ANOVA, takes two commands. First you specify the response you’re trying to model, then the control variable(s) you’re using in the fit. So to fit Figure 4 you say

MODEL mark
FIT [Fprob=yes;tprob=yes] alevel

this produces the output

***** Regression Analysis *****
Response variate: mark
   Fitted terms: Constant, alevel
*** Summary of analysis ***

<table>
<thead>
<tr>
<th></th>
<th>d.f.</th>
<th>s.s.</th>
<th>m.s.</th>
<th>v.r.</th>
<th>F</th>
<th>pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regression</td>
<td>1</td>
<td>10210.0</td>
<td>10209.6</td>
<td>58.84</td>
<td>&lt;.001</td>
<td></td>
</tr>
<tr>
<td>Residual</td>
<td>164</td>
<td>28458.0</td>
<td>173.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>165</td>
<td>38668.0</td>
<td>234.3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Percentage variance accounted for 26.0
Standard error of observations is estimated to be 13.2

*** Estimates of regression coefficients ***

<table>
<thead>
<tr>
<th></th>
<th>estimate</th>
<th>s.e.</th>
<th>t(164)</th>
<th>t pr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>14.78</td>
<td>4.96</td>
<td>2.98</td>
<td>0.003</td>
</tr>
<tr>
<td>alevel</td>
<td>1.552</td>
<td>0.202</td>
<td>7.67</td>
<td>&lt;.001</td>
</tr>
</tbody>
</table>

This tells us that the best fit is mark = 14.78 + 1.552 * alevel. The errors on the estimates of the slope and the constant are given. The ratio of the two is given as t - i.e. here the slope differs from zero by 7.67 times the estimated error, and the probability (i.e. the probability that, if the slope really were zero, such a result could arrive by chance) is, as given by Student’s t, less than 0.1%.

Now look at the analysis summary table further up. There are 166 points altogether. The total squared deviation of the values about the mean is 38668. There are 166-1 = 165 degrees of freedom so the deviation per degree of freedom is 38668/165 = 234.3. The square root of this, 15.31, is the estimate of $\sigma$ that we would get from the simple distribution: student marks vary about the mean with a standard deviation of 15.3.

After fitting the straight line, the total square deviation of measurements from the line is 28458. There are 164 degrees of freedom here (as a slope and constant have been fitted) so the mean square deviation is 28458/164=173.5, of which the square root is 13.2. So the scatter of marks about the predicted line (our estimate of $\sigma$) is 13.2.

This is better than 15.3, but not by that much. To put this another way, note that the sum of the squared deviations can be split (like was done for ANOVA) into a part due to deviations from the fit, and a part due to the spread in the control variable

$$\sum(y_i - \bar{y})^2 = \sum(y_i - (mx_i + c) + (mx_i + c) - \bar{y})^2 = \sum(y_i - (mx_i + c))^2 + \sum(mx_i + c - \bar{y})^2$$

the cross term vanishing because of the least-squares formulae for $m$ and $c$. In this case the second quantity is 10210. This is the part of the total that is explained or accounted for – we assume that variations in the control variable don’t require explanation – so 10210/38668=26% of the variation is ‘explained’ but we still don’t understand the rest.

As only one degree of freedom is at work (this is best seen by the overall count) the variance caused by scatter in $x$ feeding through to scatter in $y$ is 10210/1=10209.6, with this print format precision. The expected value of this is $\sigma^2 + m^2 V(x)$, which can be compared, using the Snedecor variance-ratio $F$ test again, with the $\sigma^2$ estimate of 173.5, to see if $m$ is meaningfully different from zero. Here the ratio is 10209.6/173.5 = 58.84 is clearly larger than one, and the probability that it could have arisen by chance is, as given, small.
The results of the regression can be displayed with \texttt{RGRAPH}. Figure 5 shows the regression of the final mark on the tutorial attendance.

It’s also important to check that you’re not missing something. The residuals, after the fit, should be normally distributed and show no systematic effects. Such checks can be done with \texttt{RCHECK}, and figure 6 shows the results of \texttt{RCHECK residuals;fitted}.

### 7.3 Multiway regression

You can model a response as a function of two control variables by extending the formula\textsuperscript{5} to be fitted: \texttt{FIT alevel+attend} will fit 3 coefficients:

\[ mark = c + m_1 \ast Alevel + m_2 \ast attendance \]

\textsuperscript{5}Note the special use of the plus symbol.
This does pretty well, accounting for 60% of the variation.

‘Interaction terms’ can be included: FIT alevel+attend+alevel.attend will fit

\[ \text{mark} = c + m_1 \times \text{Alevel} + m_2 \times \text{attendance} + m_3 \times \text{Alevel} \times \text{attendance} \]

Higher powers than straight lines can easily be included (the name ‘linear regression’ refers to linearity in the coefficients, not the variables) but this is usually a temptation which should be resisted.

Category variables (factors) can be included. FIT alevel+class will fit an overall slope for the Alevel value dependence, and an individual constant for each class. (This appears as a constant, and an extra constant for all classes other than the first). To allow a different slope for each class you add an interaction term: FIT alevel+class+alevel.class

Interpreting results needs skill. If you fit the (corrected) final mark with the lab mark you see a meaningful relationship. Likewise if you fit the mark with the attendance. If you fit it with both then you don’t get much better, because the attendance and the lab mark are so strongly correlated. You have to judge what can meaningfully be put in and taken out. To help with this, there is a statement TERMS... with which, after MODEL, you specify all the terms you might want to fit to. It then does all the appropriate summations, and you can use FIT to specify a formula and thereafter ADD and DROP to modify the formula and get results which are rapid and (more importantly) show the incremental changes.

8 Further Suggestions

These are some questions the summer school participants were invited to explore using the data provided and the Genstat package.

• We’ve seen that the marks for classes A and B differ. Is this explained by the difference in their A level scores?
• Are there any other significant differences between these two classes? Between any classes?
• The gender (male/female) of the students is also recorded. Is there a difference in their mark? If so, is this explained by any other differences?
• Students are allocated into tutorial groups of 3-4 using their A level scores to group similar students together. Show that the performance between groups varies. Is this ascribable entirely to the Alevel score differences?
• Why isn’t figure 1 Gaussian? What’s gone wrong with the Central Limit Theorem?
• What advice would you give a first year student?
• What advice would you give the teachers of first year students?

References

[2] CERNLIB, the CERN program library, is available from the CERN computer centre, CERN, Geneva
[3] NAGLIB mark 15 is available from the Numerical Algorithms Group, address as above
Overview of Mathematica

M. Trott
Wolfram Research, Inc., U.S.A.

Abstract

A general overview of Mathematica is given.

Keywords: Mathematica, conformal map, special functions, minimal surfaces, quantum well, WKB-approximation, linear turning point problems, semiclassic quantization.

Computer algebra can be described as a tool for the calculation of symbolic quantities on a computer. It begins with relatively elementary operations such as addition and multiplication of complex symbolic expressions, and includes such things as factorization of polynomials, computation of definite and indefinite integrals, solving linear and nonlinear ordinary and partial differential equations, making series expansions, residue calculations, etc. Today it is possible to carry out calculations in an hour or a few minutes on modern computers with computer algebra systems that take years to do by pencil and paper. One classical example is the calculation of the orbit of the moon, which took French astronomer Delaunay twenty years. Along with their ability to do symbolic calculations, modern computer algebra systems also include robust numerical methods, analytic properties, and numerical values for most special functions and orthogonal polynomials, as well as 2D and 3D graphics capabilities.

One of these general purpose computer algebra systems is Mathematica. This lecture gives a crude overview what Mathematica can do and what Mathematica (version 2.2) can’t do.


1 About Mathematica

1.1 Symbolic Operations

Here is an overview of the most important symbolic operations that can be done with Mathematica:

- differentiation (total and partial)
- integration (definite and indefinite)
- limits
- residues
- solving differential equations and systems of differential equations (linear and nonlinear, ordinary and partial)
- linear algebra ([including underdetermined] systems of equations, eigenvalue problems, and minimax problems)
- arbitrary powers and exponential functions of square matrices
- Taylor-, Laurent-, and Puiseux-series expansions (also for special functions)
- symbolic summation
- classical orthogonal polynomials (Hermite-, Laguerre-, Jacobi-... polynomials)
- special functions of mathematical physics (elliptic, Bessel-, Zeta-, Gamma-, Weierstrass-, hypergeometric, confluent hypergeometric functions, etc.)
- number theoretical functions on the integers
- polynomial factorization and decomposition
- computing resultants and Gröbner bases of polynomials
• variational calculations (using a package)
• solving recurrence formulae (using a package)
• abstract noncommutative operations
• pattern matching (also in commutative and associative structures)
• (symbolic) rules for functions not implemented in Mathematica can be defined by the user.

1.2 Numerical Operations

Mathematica carries out exact computations on the following sets of numbers:
• the set of complex (real) numbers (e.g. sin(π/6) and Γ(1/2))
• the integers (e.g. the 100000000th prime number)
• the rational numbers
• algebraic and irrational numbers (in a limited way)

Calculations with a finite number of digits can be done
• with machine accuracy (coprocessor accuracy allowing faster computation)
• with arbitrary accuracy (in an acceptable time with several thousand digits)
• for real numbers: interval analysis (error control)

Here is a list of the most important numerical operations that can be done with Mathematica:
• integration (also contour integrals in the complex plane)
• ordinary (also nonlinear and stiff) differential equations and systems of differential equations
• linear algebra (systems of equations and eigenvalue problems) for full matrices and, to some extent, also for sparse matrices
• classical orthogonal polynomials (Hermite-, Laguerre-, Jacobi-… polynomials)
• special functions of mathematical physics (elliptic, Bessel-, Zeta-, Gamma-, Weierstrass-, hypergeometric, confluent hypergeometric functions, etc.)
• interpolation in one and several dimensions on tensor-product grids
• sums and products, also infinite ones
• minimization problems in one and several dimensions
• zero finding in one and several dimensions
• Fourier transforms in one and several dimensions
• interval arithmetic
• pseudo-compilation of user-generated code for numerical routines

1.3 Graphics

Mathematica has the following graphics capabilities:
• Two Dimensional Graphics
  – display of given points, lines, circles, ellipsis, circular disks, polygons, and text
  – plots of continuous functions y = f(x) and of discrete functions y_i = f(x_i)
  – plots of parametric curves x = x(t), y = y(t) → y(x) multi-valued
  – plots of arbitrary multi-valued y_i = f(x_i)
  – plots of multi-valued f(x, y) = 0 (using a package)
  – graphical representation of vector fields (using a package)
• Three Dimensional Graphics
– display of given points, lines, polygons, and text
– plots of continuous functions \( z = f(x, y) \) and discrete functions \( z_i = f(x_i, y_i) \)
– plots of parametric surfaces \( x = x(s, t), y = y(s, t), z = z(s, t) \rightarrow z(x, y) \) multi-valued
– plots of parametric space curves \( x = x(t), y = y(t), z = z(t) \)
– plots of multi-valued \( f(x, y, z) = 0 \) (using a package)
– plots of arbitrary multi-valued functions given in the form \( (x_{ij}, y_{ij}, z_{ij}) \) (using a package)
– contour line plots of functions and data
– gray level and false color plots of functions and data
– graphical representation of vector fields (using a package)

• Combination of Graphics
  – inside of each other or on top of each other
  – as tables
• all graphics can be output in PostScript or Encapsulated PostScript (EPS) and machine specific bitmaps
• Animations

1.4 Programming

The following programming paradigms can be implemented with Mathematica:

• procedural
• rule-based
• functional
• object oriented

1.5 What is not yet implemented in Mathematica v.2.2

• analytically or numerically solve partial differential equations (of higher order)
• eigenvalue problems for ordinary differential equations and systems
• solve functional equations
• cylindrical algebraical decomposition
• solve integral equations
• treat Pfaff forms
• solve diophantine equations
• recognize Painlevé transcendentals as solutions of differential equations and calculate them numerically
• find arbitrary transcendental functions of matrices
• complete support of symbols (Greek letters, integral signs, etc.)
• ray-tracing, shadows and transparence in 3D graphics
• zeros of the special functions and linear combinations of them (provided, to some extent, in the package NumericalMath`BesselZeros`)
• exploitation of special properties of parameters in integration
• hypergeometric functions of several variables
  • ...

2 Basics of Mathematica as a Programming Language

2.1 General Background
Mathematica is an interactive programming system that can also be run in batch mode. To begin programming in Mathematica, type `math` for terminal versions, or click the Mathematica icon for notebook versions.

Mathematica is interactive; for example, the first input and output lines of an initial Mathematica session could look as follows: (the `In[n] :=` and `Out[n] =` are generated by Mathematica itself, and are not input by the user):

```
In[1] := 1 + 1
Out[1] = 2
In[2] := 2 + 2
Out[2] = 4
```

The basic rules for the use of Mathematica as a programming language are:

- Almost all built-in commands begin with capital letters, and are complete English words. If a command consists of several words, then the first letter of each word making up the command is also capitalized. The complete word is written without spaces, (e.g. `AxesLabel`, `ContourSmoothing`, and `TeXForm`). If the name of a person is involved (for example in the special functions of mathematical physics), the name comes first, followed immediately by the usual symbol for this function, written as a capital letter (e.g. `JacobiP`, `HermiteH`, `BesselJ`, and `RiemannZeta`). There are three classes of exceptions to this general rule. The first concerns mathematical notation, here shorter symbols are typically used—such as `E` for the number e, `I` for i, `Det` for determinant, `Sin` for sin, and `LCM` for the least common multiple. The second class includes the abbreviation `N` for numerical operations (e.g. `N` for the computation of numerical values themselves [`N[Sqrt[2]]` prints 1.41421 - not all calculated digits are displayed], `NSolve` for the numerical solution of equations, `NProduct` for . . . ), the abbreviation `D` for operations involving differentiation (e.g. `D` for differentiation, `DSolve` for solving differential equations), and the abbreviation `Q` (question) for functions that ask questions (e.g. `EvenQ` for testing if something is an even number). The third class are about 90 commands that work together with Mathematica as a programming system, and which begin with `$` (e.g. `$Language` and `$MachineType`). Mathematica knows about 800 executable commands.

- Symbols defined by the user in Mathematica usually begin with lower case letters. Variable names can be arbitrarily long, and can include both upper and lower case letters (which are distinguished from each other), `$` and numbers (but numbers can’t be used as the first character). Only complete well-developed routines should be given names starting with capital letters. Names of the form `name1_name2` are not allowed in Mathematica. Users should never introduce symbols of the form `name$` or `name$number`, because Mathematica produces symbols in this form to make names unique (see Chapter 4).

- The operation of some command can be influenced by a variety of options of the form `command -> specialValue` (e.g. `PlotPoints -> 25` and `Method -> GaussKronrod`). The possible settings for the options of a command depend on the command, and include numbers, lists, or such things as `All`, `None`, `Automatic`, `True`, `False`, `Bottom`, `Top`, `Left`, `GaussKronrod`, and `CofactorExpansion`. There are around 200 options in all.

### 2.2 Elementary Syntax in Mathematica

The algebraic operations addition, subtraction, multiplication, and division are denoted as usual by `+`, `->`, `*`, and `/`. The `*` for multiplication can be omitted by using a blank space instead. Round paranthesis `()` are used exclusively for grouping, and square brackets `[]` are used for
enclosing arguments in functions. Curly braces {} are used to enclose components of vectors and elements of sets (here arbitrarily many elements of arbitrary types are allowed, which can be nested arbitrarily often), and for iterators as range descriptions.

<table>
<thead>
<tr>
<th>Mathematical expression</th>
<th>Mathematica form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$c + b$</td>
</tr>
<tr>
<td>Subtraction</td>
<td>$d - e$</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$3x$ or $3\times x$</td>
</tr>
<tr>
<td>Division</td>
<td>$\frac{4}{y}$</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>$h^l$</td>
</tr>
<tr>
<td>Grouping with an argument</td>
<td>$f(x)$</td>
</tr>
<tr>
<td>from ... to ... discrete</td>
<td>${i, 1, 10, 1}$ or ${i, 10}$</td>
</tr>
<tr>
<td>from ... to ... continuous</td>
<td>${x, 0, 1}$</td>
</tr>
<tr>
<td>vector</td>
<td>${ax, ay, az}$</td>
</tr>
<tr>
<td>decimal number</td>
<td>3.567</td>
</tr>
<tr>
<td>string</td>
<td>&quot;a&quot;</td>
</tr>
<tr>
<td>set</td>
<td>${1, 3, a, b, c, \text{Sin}(5), &quot;iAmAString&quot;}$</td>
</tr>
</tbody>
</table>

The use of round brackets (someExpressions) for grouping and square brackets AFunction[argumentsOfAFunction] for arguments of functions is essential for correct syntax; curly brackets {} and double square brackets [[sequenceOfPositiveIntegersOr0]] are short forms for the commands List and Part.

Here is an advanced example of a typical piece of Mathematica source code. Named temporary auxiliary variables can be largely dispensed with using Mathematica’s functional programming capabilities.

```

In[2] := Show[Graphics3D[{EdgeForm[GrayLevel[0.25], Thickness[0.001]],
                       SurfaceColor[GrayLevel[0.8], RGBColor[1, 0.4, 1, 3],
                       {Polygon /@ #1, Polygon /@ #2, MapThread[Polygon[Join[#1, #2]]&,
                         {Reverse /@ Flatten[Partition[#, 2, 1]]& /0 Append[#, First[#]]& /0 #2,1],
                         Flatten[Partition[#, 2, 1]]& /0 Append[#, First[#]]& /0 #1,1]]}&00 (Map[0.92#&, #, {1}])#&
                       (Function[$, Function[$, $ + 0.8($ - $)] /0 $([Plus@ $/3]
                          ) /0 Map[1.07 - Sqrt[1.07 - (# - 0.850651)*2])&[Sqrt[ #.#]]& Nest[Flatten[Apply[Function[($1, $2, $3],
                          {{$1, $, $2}, {$3, $, $2}]}($1 + (($2 - $1)*)&$#/Sqrt[+#.#[#.#[#.[$3 - $1]]])], #, {1}], 1]]& Flatten[Function[{$}, {First[#, Plus 00 #/2, Last[$]]},
                          {Last[#, Plus 00 #/2, Last[$]]}& /0 First[$]] /0
                          (Function[Append[#, First[#]], 2, 1, Plus 00
                          #/5]]& /0 (First /0 First[Polyhedron[Polyhedron[Boxed]]
                          )], 2, 3, {$(-2)]})))], Boxed -> False];
```
3 An Example

*Mathematica* allows mathematical formulas and algorithms to be entered in a very direct way. As a small example let us implement the calculation of the series of the conformal map $w = f(z)$ after Szegö’s method [5] which maps a square in the $z$ plane onto the unit disk in the $w$ plane. The approximation of $w = f(z)$ of order $n$ is given by:

$$h_{jk} = \frac{1}{l} \int_C z^j \overline{z}^k ds$$

$$H^{(n)} = (h_{jk}) \quad j, k = 0, 1, \ldots, n$$

$$d_n = \det H^{(n)}$$

$$G^{(n)}(\xi) = \begin{cases} h_{jk}, & j = 0, 1, \ldots, n, \quad k = 0, 1, \ldots, n - 1 \\ \xi^j, & j = 0, 1, \ldots, n, \quad k = n \end{cases}$$

$$l_n(\xi) = \det G^{(n)}(\xi)$$

$$p_n(\xi) = \frac{l_n(\xi)}{\sqrt{d_{n-1} d_n}}$$

$$p_0(\xi) = 1$$

$$k_n(\alpha, \beta) = \sum_{i=0}^{n} p_n(\alpha) p_n(\beta)$$

$$w_n(z) = \frac{\pi}{4k_n(0, 0)} \int_0^z k_n(0, \xi)^2 d\xi$$

Here $l$ is the length of the boundary of the square; the integration has to be carried out along the boundary of the square. The $p_n(\xi)$ form orthogonal polynomials (see also exercise 4 of Ch. 13). $H^{(n)}$ and $G^{(n)}$ are square matrices of dimension $n$ with elements $h_{jk}, g_{jk}$ respectively. Here the method implemented. The order in $z$ of the series is $\operatorname{ord}$ and $\operatorname{var}$ stands for $z$. 
ConformalMapSquareToUnitDisk[ord_Integer?Positive, z_] :=
Module[h, H, d, l, p, k, t, a, b, boundaryLength, integrand,
  edgeList = {-1 + I, 1 + I, 1 - I, -1 - I},
  lineSegments = Partition[Append[edgeList, First[edgeList]], 2, 1],
  boundaryLength = (Plus @@ (Abs[#[[2]] - #[[1]]] & /@ lineSegments)) /.
  (* scalar product *)
  c_Complex -> Conjugate[c],
  (* integrand *)
  integrand[j_, k_] = Plus @@ (Abs[#/2 - #/1] & /@ lineSegments) /
  (* scalar product *)
  c_Complex -> Conjugate[c]) & /@ lineSegments);

(* h[j_, k_] := h[j, k] = 1/boundaryLength Integrate[integrand[j, k], {t, 0, 1}];
  (* gram determinants *)
  H[n_] := Array[h, {n + 1, n + 1}, 0];
  d[n_] := d[n] = Det[H[n]];
  G[n_, xi_] := Array[If[#2 < n, h[#1, #2], xi^#1] &, {n + 1, n + 1}, 0];
  l[n_, xi_] := l[n, xi] = Det[G[n, xi]];
  (* Szegoe polynomials *)
  p[0, xi_] = 1;
  p[n_, xi_] := p[n, x] = l[n, xi]/Sqrt[d[n] d[n - 1]];
  (* Szegoe kernel *)
  k[a_, b_] = Sum[p[i, a] p[i, b], {i, 0, ord}];
  Cancel[Pi/(4 k[0, 0]) Expand[Integrate[k[0, xi]^2, {xi, 0, z}]]]]

Here an example:

In[4] := ConformalMapSquareToUnitDisk[8, z]

Out[4] = (Pi (10407280578566400 z + 750935631333888 z^5 + 27542148640864 z^9 -
  1259495965728 z^{13} + 11641881537 z^{17}) / 35298905177849856

Using Mathematica graphics cababilities we can easily visualize the conformal map
generated by the last function. (The left picture shows a mesh in the square with the corners
-1 + i, 1 + i, 1 - i, -1 - i and the right picture shows the mesh after mapping; the unit disk
is shown underlying in gray.)

  points = Table[x + I y, {x, -1., 1., 1/pp}, {y, -1., 1., 1/pp}];
  Show[GraphicsArray[
    Graphics[{Thickness[0.001], Line /@ #, Line /@ Transpose[#]}&[Map[{Re[#], Im[#]} &, points, {-1}]],
    AspectRatio -> Automatic, PlotLabel -> "z - plane",
    PlotRange -> {{-1.2, 1.2}, {-1.2, 1.2}}],
    Graphics[{GrayLevel[3/4], Disk[{0, 0}, 1]},
    Thickness[0.001], Line /@ #, Line /@ Transpose[#]}&[Map[{Re[#], Im[#]} &,
      Map[Function[z, Evaluate[N[z]]], points, {-1}], {-1}]],
    AspectRatio -> Automatic, PlotLabel -> "w - plane",
    PlotRange -> {{-1.2, 1.2}, {-1.2, 1.2}}],
    Frame -> True]};
4 What Computer Algebra and Mathematica Can and Cannot Do For You

Without further comment, we include the following four quotes concerning whether or not Mathematica (or more generally computer algebra) is a useful tool for solving your concrete problems. Clearly, this cannot be decided independent of the problem. It is clear, however, that for many applications in mathematics, in the applied sciences, in engineering, in finance and in other fields computer algebra is a very useful tool.

- Two goals for PSE’s [problem solving environments, including computer algebra systems] are, first, that they enable more people to solve more problems more rapidly, and second, that they enable many people to do things that they could not otherwise do.

  from [4]

- The impact on mathematics of computer algebra and other forms of symbolic computing will be even larger than the impact of numeric computing has been.

  from [1]

But:

- Computer algebra is no substitute for mathematical creativity and mathematical knowledge; consequently, it is surely no universal mathematical problem solver. However, it makes the use of mathematical knowledge easier.

  from the definition of “computer algebra”;
special interest group of GI, DMV, GAMM
Of course, just as with paper and pencil calculations, the course of the evaluation [with a computer algebra system] must be guided with ingenuity and cleverness by the human mind behind the calculation.

But nevertheless:

... let us enjoy the present exciting transition era, where we can both enjoy the rich human heritage of the past, and at the same time witness the first crude harbingers of the of the marvelous computer-mathematics revolution of the late 21st century.

Parts of this lecture are taken from my forthcoming book "The Mathematica Guidebook", TELOS 1995, with permission of Springer-Verlag.

Bibliography


Special Functions in Mathematica

Abstract

Some examples for the effective use of special functions in connection with Mathematica are presented.

The aim of this lecture is to introduce the nomenclature, to illustrate the power of Mathematica as a tool for dealing with special functions, and to show how it can be used to provide effective solutions to a variety of problems.

Never be intimidated by special functions and their appearance in Mathematica output from integration. (Forthcoming versions of Mathematica will have special functions not discussed in Abramowitz/Stegun [1].) The special functions are extremely useful tools for obtaining closed form as well as series solutions to a variety of problems arising in science.
and engineering. Mathematica is able to use them efficient, it can calculate numerical values for them, make series solutions, differentiate and integrate them, and can simplify expressions involving special functions by using special identities known for these functions.

1 Some Important Classes of Special Functions implemented in Mathematica

We briefly describe some important classes of special functions implemented in Mathematica.

There are many other special functions present, that we do not mention here.

1.1 Hypergeometric Functions

Gauss’ hypergeometric function \( _2F_1(a, b; c; z) \) is defined by

\[
_2F_1(a, b; c; z) = \sum_{k=0}^{\infty} \frac{(a)_k (b)_k z^k}{(c)_k k!},
\]

where \((d)_k\) is the Pochhammer symbol. As an immediate generalization of this function, we get the generalized hypergeometric function \( _pF_q(a_1, a_2, \ldots, a_p; b_1, b_2, \ldots, b_q; z) \):

\[
_pF_q(a_1, a_2, \ldots, a_p; b_1, b_2, \ldots, b_q; z) = \sum_{k=0}^{\infty} \frac{(a_1)_k (a_2)_k \ldots (a_p)_k z^k}{(b_1)_k (b_2)_k \ldots (b_q)_k k!}.
\]

We first examine the Mathematica formulae for these functions:

<table>
<thead>
<tr>
<th>Hypergeometric2F1[a, b, c, z]</th>
<th>represents the Gauss hypergeometric function ( _2F_1(a, b; c; z) ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>HypergeometricPFQ[{a_1, a_2, \ldots, a_p}, {b_1, b_2, \ldots, b_q}, z]</td>
<td>represents the generalized hypergeometric function ( _pF_q(a_1, a_2, \ldots, a_p; b_1, b_2, \ldots, b_q; z) ).</td>
</tr>
</tbody>
</table>

Many functions arising in daily computations can be expressed in terms of hypergeometric functions. For instance the following gives the partial sums of the Taylor expansion of cos:

\[
\text{In}[1]:= \cos(x) + (-1)^n x (2n + 2) \star \text{HypergeometricPFQ}[\{1\}, \{n + 3/2, n + 2\}, -x^2/4]/(2n + 2)!
\]

\[
\text{Out}[1]= \cos(x) + \frac{(-1)^n x^2 + 2 n \text{HypergeometricPFQ}[\{1\}, \{3/2 + n, 2 + n\}, -x^2/4]}{(2 + 2 n)!}
\]

\[
\text{In}[2]:= \text{Expand}[\text{PowerExpand}[\% \, / . \, n \rightarrow 6]]
\]

\[
\text{Out}[2]= 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \frac{x^8}{40320} - \frac{x^{10}}{3628800} + \frac{x^{12}}{479001600}
\]

\[
\text{In}[3]:= \text{Normal}[	ext{Series}[\cos(x), \{x, 0, 12\}]]
\]

\[
\text{Out}[3]= 1 - \frac{x^2}{2} + \frac{x^4}{24} - \frac{x^6}{720} + \frac{x^8}{40320} - \frac{x^{10}}{3628800} + \frac{x^{12}}{479001600}
\]
Many related quantities can be expressed in hypergeometric functions — for instance the closed form of the Padé approximation of degree $(m,n)$ of elementary functions, polynomials of degree five can be solved in closed form with hypergeometric functions.

Again the iterated integral of $\exp(-z^2)$ can be expressed as a generalized hypergeometric function. We express $\exp(-z^2)$ by its series representation, integrate each term $m$ times and obtain the following series, which Mathematica can sum up in closed form:

\[
\text{iteratedErfIntegral}[m, z] = \sum \frac{z^m}{\Gamma[1 + m]}
\]

Here is a random numerical check of the result. (Unfortunately Mathematica simplifies iteratedErfIntegral for integer $m$ to Laguerre functions, so that the following is not automatically simplified symbolically to zero):

1.2 Elliptic Integrals and Elliptic Functions

Elliptic integrals arise in the integration of expressions of the form

\[
R(t, \sqrt{a_3 t^3 + a_2 t^2 + a_1 t + a_0}) \quad \text{and} \quad R(t, \sqrt{a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0}),
\]

where $R(x,y)$ is a rational function. Here an example:

The Mathematica commands for elliptic integrals are EllipticF[$\varphi$, $m$], EllipticE[$\varphi$, $m$] and EllipticPi[$n$, $\varphi$, $m$].

Elliptic functions are the inverse functions of elliptic integrals. They are trigonometric functions of the so called Jacobi amplitude:

\[
\text{JacobiAmplitude}[u, m] \quad \text{represents the Jacobi amplitude am}(u|m).
\]

With the elliptic function JacobiAmplitude and the elliptic integral EllipticF we can make a picture of a very interesting minimal surface: one which has six pairwise parallel edges of a cube as its boundary. The equation of this minimal surface is $h(x)h(y) = h(z)$ where $h(x)$ is...
the inverse function of \(x(h)\). \(x(h)\) is defined via

\[
x(h) = \int_0^h \frac{1}{\sqrt{1 + t^2 + t^4}} dt
\]

([4]). After writing the above integral in terms of \(\text{EllipticF}\) and inverting the relation we have:

\[
In[9] := \text{Module}[\{f, g, h, \text{eps} = 10^{-14}, \text{pp} = 60\},
\]

\[
f[t_] := \text{If}\[t < 1, 0, \text{Pi}\] - \text{ArcTan}\[-\text{Sqrt}[3](1 + t)/(1 - t)];
\]

(* the above integral *)

\[
g[t_] := 2/3N[\text{EllipticF}[f[t], 8/9] - \text{EllipticF}[\text{Pi}/3, 8/9]];\]

(* the function \(h\) after inversion *)

\[
h[z_] := N[(\text{Tan}[\#] - \text{Sqrt}[3])/(\text{Tan}[\#] + \text{Sqrt}[3]) & [
\]

\[
\text{If}[\#, N[\text{Pi}/2], \#, \# - \text{Pi}]] & \text{Chop}[N[\text{JacobiAmplitude}[3/2z + \text{EllipticF}[\text{Pi}/3, 8/9], 8/9], 10^{-7}])];\]

\[
\text{Show}[\text{Graphics3D}[\{\text{EdgeForm}[\{\text{Thickness}[0.001], \text{GrayLevel}[0.8]\}],
\]

\[
\text{Graphics3D}[\text{Plot3D}[g[h] \ b[y]],
\]

(* starting and ending slightly away from the edges of the cube *)

\[
\{x, \text{eps, } \#, - \text{eps}, \{y, \text{eps, } \#, - \text{eps}\},
\]

\[
\text{PlotPoints} \rightarrow \text{pp}, \text{DisplayFunction} \rightarrow \text{Identity}]][[1]],
\]

\[
\{\text{Thickness}[0.01], \text{GrayLevel}[0],
\]

(* The edges of the cube which are touched *)

\[
\text{Line}[\{\{0, 0, 0\}, \{\#, 0, 0\}\}], \text{Line}[\{\{0, 0, 0\}, \{0, \#, 0\}\}],
\]

\[
\text{Line}[\{\#, 0, 0\}, \{\#, \#, \}\}], \text{Line}[\{\{0, \#, \#\}, \{\#, \#, \\}\}],
\]

\[
\text{Line}[\{\#, 0, 0\}, \{\#, 0, \#\}], \text{Line}[\{\{0, 0, \#\}, \{0, \#, \#\}\}],
\]

\[
\{\text{Thickness}[0.01], \text{GrayLevel}[0], \text{Dashing}[0.02, 0.02],
\]

(* the other edges of the cube dashed*)

\[
\text{Line}[\{0, 0, 0\}, \{0, 0, \#\}], \text{Line}[\{\#, \#, 0\}, \{\#, \#, \}\}],
\]

\[
\text{Line}[\{\#, 0, 0\}, \{\#, \#, 0\}], \text{Line}[\{0, \#, 0\}, \{0, \#, \#\}],
\]

\[
\text{Line}[\{0, 0, \#\}, \{\#, 0, \#\}], \text{Line}[\{0, 0, \#\}, \{0, \#, \#\}]]],
\]

\[
\text{Axes} \rightarrow \text{False}, \text{Boxed} \rightarrow \text{False}, \text{ViewPoint} \rightarrow \{1, -2, 0.9\},
\]

\[
\text{PlotRange} \rightarrow \text{All}, \text{BoxRatios} \rightarrow \{1, 1, 1\}] & (* the boundary coordinates *)
\]

\[
2/3 (\text{EllipticF}[2\text{Pi}/3, 8/9] - \text{EllipticF}[\text{Pi}/3, 8/9]) // N; \]
Jacobi’s elliptic function also play an important role for the closed form solution of many electrostatic problems.

2 Two examples from physics for the use of Special Functions

2.1 Rocket with discrete propulsion

The discrete rocket problem is as follows. Imagine a rocket which burns its fuel in discrete masses of size \((m_i - m_f)/n\) (where \(m_i\) is the initial mass, \(m_f\) the final mass, and \(n\) is the number of burns) rather than continuously. We will derive an analytic formula for the final velocity as a function of \(n\). What happens in the limiting case \(n \to \infty\)?

One easily sees from the conservation of momentum law that the final velocity \(v_f\) is

\[
v_f = \frac{u}{n} \left( \frac{m_i - m_f}{n} \sum_{k=1}^{n} \frac{1}{m_f + k(m_i - m_f)/n} \right).
\]

To find an analytical expression for this sum as a function of \(n\), we use the package Algebra\`SymbolicSum\`.

In[1] := Needs["Algebra\'SymbolicSum\"]

In[2] := vf = SymbolicSum[(mi - mf)/n \[ \times \] 1/(mf + k (mi - mf)/n), (k, n)]

Out[2] = u (-PolyGamma[0, 1 + \[ \frac{\[ \text{mf}\]}{\text{mi}} \] ] + PolyGamma[0, 1 + n + \[ \frac{\[ \text{mf}\]}{\text{mi}} \] ])

Substituting \(m_i/m_f \to \nu\), and normalizing the nozzle velocity, we get a unitless result:

In[3] := vf = (vf /. mf/(mf + mi) \[ \rightarrow \] 1/(1 + nu))/u

Out[3] = -PolyGamma[0, 1 + \[ \frac{n}{\text{-1 + nu}} \] ] + PolyGamma[0, 1 + n + \[ \frac{n}{\text{-1 + nu}} \] ]

In order to find the limiting value for an infinite number of impulses, we make use of the package Calculus\`Limit\`:

In[4] := Needs["Calculus\'Limit\"]
In[5] := Clear[\text{nu}]; \text{Limit}[\nu f, n \rightarrow \text{Infinity}]

\text{Out[5]} = \text{Log}[\nu + 1] + \text{Log}[\frac{\nu}{1 + \nu}]

which is the known result [3].

Taking account that the masses are positive, we get

\text{In[6]} := \% / . \text{Log}[a_, n] \rightarrow \text{Log}[a \; b]

\text{Out[6]} = \text{Log}[\nu]

or with units: \( v_f = \nu \ln(m_i/m_f) \).

### 2.2 Perturbation Theory

Consider the following eigenvalue problem (where \( \lambda \) is the eigenvalue):

\[-y''(x) + \alpha x y(x) = \lambda y(x), \quad y(-L/2) = y(L/2) = 0.\]

For small \( \alpha (\alpha << \pi/L^2) \), starting with the exact solution of the problem, determine the dependence of \( \lambda_i = \lambda_i(\alpha) \) on \( \alpha \) in the form

\[\lambda_i(\alpha) \approx \lambda_i(\alpha = 0) + c_1\alpha + c_2\alpha^2.\]

First one may think of second order time-independent perturbation theory

\[c_{i2} = \sum_{j=1}^{\infty} \frac{\left| \int_{-L/2}^{L/2} y_i(x, \alpha = 0) x y_j(x, \alpha = 0) \right|^2}{\lambda_i(\alpha = 0) - \lambda_j(\alpha = 0)}\]

but this is a lecture on special functions, so we don’t use the above formula. Such an approach is difficult to generalize to eigenvalue problems with continuous spectra, e.g. to the following interesting generalization of the problem (and an important one in practical applications):

\[-y''(x) + \alpha x y(x) + V_0 \Theta(x - L^2) = \lambda y(x), \quad y(-L/2) = y(L/2) = 0.\]

We begin with the exact solution of the differential equation for arbitrary \( \alpha \):

\text{In[1]} := \text{DSolve}[\{-y''[x] + \alpha x y[x] == \lambda y[x], y[x], x, \text{DSolveConstants} \rightarrow b\}]

\text{Out[1]} = \{\{y[x] \rightarrow \text{AiryBi}[a^{1/3} \left( \frac{1}{a} - \frac{1}{\lambda} \right) + x] \; b[1] + \text{AiryAi}[a^{1/3} \left( \frac{1}{a} - \frac{1}{\lambda} \right) + x] \; b[2]\}\}

The coefficients \( b[1], b[2] \) are determined from the boundary condition at \( x = \pm L/2 \).

\[b_2 \text{Ai} \left( \frac{-\lambda + \frac{2L}{\alpha^{2/3}}}{\alpha^{2/3}} \right) + b_1 \text{Bi} \left( \frac{-\lambda + \frac{2L}{\alpha^{2/3}}}{\alpha^{2/3}} \right) = 0\]

\[b_2 \text{Ai} \left( \frac{-\lambda - \frac{2L}{\alpha^{2/3}}}{\alpha^{2/3}} \right) + b_1 \text{Bi} \left( \frac{-\lambda - \frac{2L}{\alpha^{2/3}}}{\alpha^{2/3}} \right) = 0\]

This linear system of equations in \( b_1, b_2 \) has a nontrivial solution only if the associated coefficient determinant \( \text{determin} \) vanishes.

\text{In[2]} := s1[x_] = \text{AiryAi}[\left( -1 + a \; x \right)/a^{\left( 2/3 \right)}];

\text{s2[x_] = AiryBi[\left( -1 + a \; x \right)/a^{\left( 2/3 \right)}];}
At this point it remains only to develop a series expansion for the solution $\lambda_i(\alpha)$ of this equation. We still don’t need an explicit solution, since

$$\lambda_i(\alpha) \approx \lambda_i(\alpha = 0) + c_1 \alpha + c_2 \alpha^2 + \cdots$$

$$\approx \lambda_i(\alpha = 0) + \frac{\partial \lambda_i(\alpha)}{\partial \alpha} \bigg|_{\alpha=0} \alpha + \frac{1}{2!} \frac{\partial^2 \lambda_i(\alpha)}{\partial \alpha^2} \bigg|_{\alpha=0} \alpha^2 + \cdots$$

and the second derivative of $\lambda_i(\alpha)$ with respect to $\alpha$ can be obtained immediately by implicit differentiation of the exact eigenvalue equation:

$$\lambda_i(\alpha) \approx \lambda_i(\alpha = 0) + c_1 \alpha + c_2 \alpha^2 + \cdots$$

All of the derivatives involved can be found directly by differentiating the remaining $\text{lhs}[\text{a}, \alpha]$ ($= \text{deter}$), or alternatively, from

We go back to $\text{deter}$, and look more carefully at its behavior for $\alpha \to 0$. Because the $\lambda_i(\alpha = 0)$ are finite and positive, all variables in the Airy function go to $00$ as $\alpha \to 0$. We now make use of the first two terms of the asymptotic formula.

In[7] := ord1 = 2;

asymptoticRules =
{\text{AiryAi}[x_\_] -> Normal[Series[AiryAi[x], \{x, -\text{Infinity}, ord1\}]],
\text{AiryBi}[x_\_] -> Normal[Series[AiryBi[x], \{x, -\text{Infinity}, ord1\}]]};

In the interest of space, we suppress some output, and name the intermediate results. (For this kind of calculation, protecting the variables is usually not necessary; we do it interactively just once.) Substituting asymptoticRules in deter gives

Since this expression is zero, we should eliminate the denominator:
The products of the angle functions oscillate infinitely in the limiting case \( \alpha \to 0 \). Hence, we rewrite the angle functions with a single variable in order to extract a non-oscillating part.

We again expand the arguments of the resulting trigonometric functions in a series.

Next we simplify the roots (without loss of generality \( \alpha > 0 \)):

Finally, we perform additional simplifications:
This is our simplified result for $\text{deter}$. First we consider the limiting case $\alpha \to 0$:}

\text{In[19]:= Needs["Calculus\`Limit\"]}

\text{In[20]:= Limit[deterAsymp, a \to 0, Direction \to -1]}

\text{Out[20]= \(-2304 \sqrt{2} \cdot 1^2 \sin[\sqrt{2}] \cdot L\)}

Thus $\sqrt{\lambda} = j\pi, j = 1, 2, \ldots$, which can also be seen more or less directly from the differential equation for $\alpha = 0$. Now we turn to the computation of $c_{i1}, c_{i2}$. Since

\text{In[21]:= D[deterAsymp, a] /. a \to 0}

\text{Out[21]= 0}

we have $c_{i1} = 0$, and the computation of $c_{i2}$ simplifies to:

\text{In[22]:= \((-1/2 \cdot D[deterAsymp, \{a, 2\}] / D[deterAsymp, 1]) /. a \to 0\)}

\text{Out[22]= \(-720 \sqrt{2} \cdot L \cos[\sqrt{2} \cdot L] + 48 \sqrt{2} \cdot \sqrt{1} \cdot L \cdot \cos[\sqrt{2} \cdot L] - 50 \sqrt{2} \cdot \sin[\sqrt{2} \cdot L] + 1152 \sqrt{2} \cdot \sqrt{1} \cdot L \cdot \sin[\sqrt{2} \cdot L]) / (2 \cdot(13/2 \cdot \sqrt{2} \cdot L \cos[\sqrt{2} \cdot L] - 4608 \sqrt{2} \cdot \sqrt{1} \cdot L \cdot \sin[\sqrt{2} \cdot L]))\)}

We now take account of the eigenvalue equation for $\alpha = 0$,

\text{In[23]:= \% /. \{\sin[L \cdot \sqrt{1}] \to 0\} // Simplify}

\text{Out[23]= \(-\frac{15 + 1 \cdot L^2}{48 \cdot 1^2}\)}

so that for the eigenvalue of $\alpha = 0$, we finally obtain

\text{In[24]:= \% /. L \to (i \cdot \pi / L)^2}

\text{Out[24]= \frac{L^4 \cdot (-15 + \pi^2 \cdot i^2)}{48 \cdot \pi^4} \cdot \frac{L^2}{i^2}}

Increasing $\text{ord1, ord2 and ord3}$ and calculating the corresponding expressions for higher order corrections.

For the relevance of such calculations to solid state physics, see [2].

Parts of this lecture are taken from my forthcoming book "The Mathematica Guidebook", TELOS 1995, with permission of Springer-Verlag.

\textbf{Bibliography}
WKB Approximation

Abstract

A Mathematica program for the derivation of high order WKB quantization rules is developed.

The aim of this lecture is to show a more advanced symbolic computation using Mathematica. First approximations of the Schrödinger equation near a linear turning point are studied and second higher order WKB approximations are studied.

1 Uniform approximation of linear turning point problems

In applications one often encounters the following differential equation (time-independent Schrödinger equation):

\[ y''(x) = \lambda^2 f(x) y(x) \]

where \( \lambda^2 \) is a large parameter and \( f(x) \) a real valued function. A typical example for this type of equation is the Schrödinger equation. In many cases of interest for a given \( f(x) \) an exact solution can not be found and one has to use approximative methods. A very popular method is the so-called WKB approximation (see [10], [6], [4], [3]). It consists of the following: We change the independent and dependent variable via:

\[ z(x) = \int^{x} \sqrt{f(x)} \, dx \quad w(z) = f^{\frac{1}{4}}(x) y(x) \]

Then the differential equation in \( w(z) \) is:

\[ w''(z) = \left( \lambda^2 - f^{-\frac{3}{4}}(x(z)) \frac{d^2}{dx^2} f^{-\frac{1}{4}}(x(z)) \right) w(z) \]

Neglecting the second term we get the following two linearly independent solutions:

\[ y_1(x) = f^{-\frac{1}{4}}(x) \exp \left( \pm \lambda \int^{x} \sqrt{f(x)} \, dx \right) \]

One sees that neglecting of the second term is not possible if \( \lambda^2 \) is not large or if \( x \) is near a zero of \( f(x) \). In the case of a simple zero of \( f(x) \) (the so-called linear turning point problem) a similar change of variable

\[ \frac{2}{3} z^\frac{1}{2}(x) = \int^{x} \sqrt{f(x)} \, dx \quad w(z) = \left( \frac{f(x)}{z(x)} \right)^{\frac{1}{4}} y(x) \]

yields
Again neglecting the second term on the right hand side we obtain the following two solutions:

\[
y_1(x) = \left( \frac{z(x)}{f(x)} \right)^{\frac{1}{4}} \text{Ai} \left( \lambda \frac{2}{3} \int x^f(x) \, dx \right)
\]

\[
y_2(x) = \left( \frac{z(x)}{f(x)} \right)^{\frac{1}{4}} \text{Bi} \left( \lambda \frac{2}{3} \int x^f(x) \, dx \right)
\]

To make the solutions sensible also at the turning points, we modify the transformation from \( x \) to \( z \) in the following way: If \( x_0 \) is a zero of \( f(x) \), and (without loss of generality) assuming \( f(x > x_0) > 0 \), we choose:

\[
\frac{2}{3} z^3(x) = \int_{x_0}^x \sqrt{f(x)} \, dx \quad x \geq x_0
\]

\[
\frac{2}{3} (-z)^3(x) = \int_{x_0}^x \sqrt{-f(x)} \, dx \quad x \leq x_0
\]

For large \( \lambda^2 \), this is a nondivergent solution also around \( x_0 \). Using the asymptotic expansions for the Airy functions, the WKB solutions are the limit of appropriate linear combinations of these solutions.

As an example let us investigate the well-known harmonic oscillator as an eigenvalue problem (we set \( \lambda = 1 \), which can be always be done, after a suitable change of variables). The straightforward change of variables gives:

\[
y''(x) = (\lambda^2 - \varepsilon) y(x)
\]

The normalized solutions which go to zero as \( x \) goes to infinity are:

\[
\varepsilon_n = 2n + 1
\]

\[
y_n(x) = \frac{1}{\sqrt{\sqrt{\pi} 2^n n!}} e^{-\frac{x^2}{2}} H_n(x)
\]

where \( n = 0, 1, 2, \ldots \) and \( H_n(x) \) are the Hermite polynomials. We implement this:

\[
\text{exactSolutionHO}[n_, x_] := \frac{1}{\sqrt{\sqrt{\pi} 2^n n!}} e^{-\frac{x^2}{2}} H_n(x)
\]

Let us now investigate the uniform approximation near the right turning point \( x_0 = \sqrt{\varepsilon} \).

After carrying out the relevant integrations needed in \( z = z(x) \)

\[
\text{gr} = \text{Integrate}[[\text{Sqrt}[\text{Pi} 2n n!] \text{Exp}[-x^2/2] \text{HermiteH}[n, x]]]
\]

\[
\text{Out}[4] = \frac{\text{Log}[	ext{Sqrt}[\text{e}]] + \frac{\text{Sqrt}[-e + x^2]}{2} - \text{Log}[x + \text{Sqrt}[-e + x^2]]}{2}
\]
we can also implement the uniform approximations:

\[
\text{In[6]:=} \quad \begin{align*}
& \xi_{\pm}, x_{\pm} := (3/2)^{2/3} \left( (\log[\sqrt{e}])^{2/3} \right) + (x^{2} - e)^{2/3} \left( (1/2) \right) + \arctan\left( \sqrt{e} \right) - (x^{2} - e)^{-1} + (x^{2} - e) \right) / x; \quad x > \sqrt{e} \\
& \xi_{\pm}, x_{\pm} := (-3/2)^{2/3} \left( (\log[\sqrt{e}])^{2/3} \right) + (x^{2} - e)^{2/3} \left( (1/2) \right) + \arctan\left( \sqrt{e} \right) - (x^{2} - e)^{-1} + (x^{2} - e) \right) / x; \quad x < \sqrt{e}
\end{align*}
\]

To fulfill the boundary condition \( y(x \to \infty) = 0 \) we have to choose the solution containing the Airy function \( \text{Ai}(x) \):

\[
\text{In[8]:=} \quad \text{uniformApproximationHO}[n, x_] := (\xi[2n + 1, x] / f[2n + 1, x])^{(1/4)} \text{AiryAi}[\xi[2n + 1, x]]
\]

The expression \( z(x)/f(x) \), which appeared in the neglected term of the above differential equation, remains finite as \( x \) approaches \( x_{0} \):

\[
\text{In[9]:=} \quad \text{Simplify[Normal[Series[((3/2)^{(2/3)} / f[e, x])^{(1/4)},}
\quad \text{\{x, Sqrt[e], 0\}]]) // PowerExpand}
\]

Finally, for comparison purposes let us implement the WKB solutions. We get them by the expansion of the Airy function for large positive and negative arguments:

\[
\text{In[11]:=} \quad \text{asympRight}[x_] = \text{Series}[\text{AiryAi}[x], \{x, \infty, 1\}] // \text{Normal};
\quad \text{asympLeft}[x_] = \text{Series}[\text{AiryAi}[x], \{-\infty, 1\}] // \text{Normal};
\]

\[
\text{WKBsolHO}[n, x_] := (\xi[2n + 1, x] / f[2n + 1, x])^{(1/4)} * \\
\quad \text{If}[x > \sqrt{2n + 1}, \text{asympRight}[\xi[2n + 1, x]],
\quad \text{asympLeft}[\xi[2n + 1, x]]]
\]

Now let us graphically show the various solutions (we use a form which mostly used in quantum mechanics textbooks):
In[14]:= With[{nmax = 5},
Show[Graphics[
(* the potential x^2 *)
{Thickness[0.01], Plot[x^2, {x, -1.1 Sqrt[2nmax + 1], 1.1 Sqrt[2nmax + 1]},
  DisplayFunction -> Identity][[1, 1, 1]],
{Thickness[0.003], GrayLevel[0.2], Dashing[{0.002, 0.002}],
(* the scaled WKB solutions, not shown to
  the left of the left turning point *)
Table[Plot[1/scaling[n] WKBsolHO[n, x] + (2n + 1),
  {x, -0.99 Sqrt[2n + 1], 0.99 Sqrt[2n + 1]},
  PlotPoints -> 60, DisplayFunction -> Identity][[1, 1, 1]],
  {n, 0, nmax}],
Table[Plot[1/scaling[n] WKBsolHO[n, x] + (2n + 1),
  {x, 1.01 Sqrt[2n + 1], 1.6 Sqrt[2n + 1]},
  PlotPoints -> 60, DisplayFunction -> Identity][[1, 1, 1]],
  {n, 0, nmax}],
(* the energy eigenvalue lines *)
{Thickness[0.007], GrayLevel[0.8],
Table[Line[{{-1.8 Sqrt[2n + 1], 2n + 1},
  { 1.8 Sqrt[2n + 1], 2n + 1}}], {n, 0, nmax}}],
(* the exact solutions *)
{Thickness[0.001],
Table[Plot[exactSolutionHO[n, x] + (2n + 1),
  {x, -1.8 Sqrt[2n + 1], 1.8 Sqrt[2n + 1]},
  PlotPoints -> 60, DisplayFunction -> Identity][[1, 1, 1]],
  {n, 0, nmax}]},
(* the scaled uniform approximation solutions, not shown to
  the left of the left turning point *)
{Thickness[0.001], GrayLevel[0.5], Dashing[{0.01, 0.01}],
Table[Plot[1/scaling[n] uniformApproximationHO[n, x] + (2n + 1),
  {x, -0.99 Sqrt[2n + 1], 1.8 Sqrt[2n + 1]},
  PlotPoints -> 60, DisplayFunction -> Identity][[1, 1, 1]],
  {n, 0, nmax}]}
], Frame -> True, AspectRatio -> 1.2, FrameLabel -> None];]

The WKB solutions diverge at the right turning point and are not useful approximations there, while the uniform approximation coincide to the accuracy of the plot with the exact solution, including at the turning point itself as long as we stay away from the second turning point x0 = -Sqrt[e].
2 WKB Approximations of Higher Order

An approximate solution for the eigenvalue of the one-dimensional Schrödinger equation (here $m = 1$, and $\bar{\hbar} = \hbar/(2\pi)$ explicitly stand for the derivation of the relevant equations determined by a "small" parameter) (the so-called WKB approximation).

$$\left(-\frac{\hbar^2}{2} \frac{d^2}{dx^2} + V(x) - E\right) \psi(x) = 0.$$

We restrict ourselves here to the case where the potential has just one minimum. Assuming a solution of the form

$$\psi(x) = \exp\left(i\hbar \int_x^\infty \sum_{n=0}^\infty \left(\frac{\hbar}{i}\right)^n S_n(x) \, dx\right)$$

for the Schrödinger equation, we get the following recurrence formula for $S_n(x)$ ([9]):

$$S_0(x) = \sqrt{E - V(x)}$$

$$S'_{n-1}(x) = \sum_{m=0}^n S_{n-m}(x)S_m(x)$$

Now interpreting $x$ as a complex variable and $V(x)$ as a complex function, we are led to the following condition for the eigenvalue $E$:

$$\int S_0(x)dx + \sum_{n=1}^\infty \left(\frac{\hbar}{i}\right)^n \int S_n(x)dx = \left(n + \frac{1}{2}\right)\hbar$$

Thus, $E_j (j = 0, 1, \ldots)$ is an eigenvalue for the above differential equation if and only if this equation is satisfied. The $x$-integrals are taken along a path in the complex $x$-plane which contains both of the classical turning points $x_1$ and $x_2$. Both $x_1$ and $x_2$ are solutions of the equation $V(x) = E$.

Frequently, the potential $V(x)$ is given only for real numbers $x$ (perhaps from a numerical calculation). Then the above integral formula is of little use. Even if we put the integration path on the real axis, for $n > 1$, in general $S_n(x)$ has a nonintegrable singularity. Moreover, the integrands which arise can be greatly simplified by integration by parts.

One way out of these difficulties is to use integration by parts with respect to $x$, and to reduce the strongly singular integrands to integrable integrands using integration by parts over $E$. (For details, see [8], [5], [2], [1], [3], [7])

We will implement the computation of these reduced forms of the integrals over the $S_n(x)$ for $n = 1$ to 8. To this end, first find $S_n(x)$, and then integrate the parts containing $V'(x)$ as often as possible by parts. Furtheron we will integrate the terms so obtained as often as possible by parts, without increasing the maximum singularity in the denominator. Finally, we integrate the expressions thus obtained by parts with respect to $E$ and write the results in an appropriate form.

Since arbitrary complete differentials can be added to the expressions of different orders, the expressions to be found are not uniquely defined. Here we implement our own version instead of strictly following [2]. We begin with the computation of $S_0(x)$. We separate the definition into even and odd indices, since for even indices, the term $S^2_{k}(x)$ does not appear. Since the higher terms of $S_n(x)$ are required several times in the definition, we save them.
In[1]:= S[0] = Sqrt[e - v[x]];

S[k_?EvenQ] := S[k] = Expand[(-1/2 D[S[k - 1], x] -
      Sum[S[m] S[k - m, {m, 1, k/2}] + 1/2 S[k/2]^2]/S[0]]

S[k_?OddQ] := S[k] = Expand[(-1/2 D[S[k - 1], x] -
      Sum[S[m] S[k - m, {m, 1, (k - 1)/2}]]/S[0]]

Now we integrate the terms containing \( V'(x) \). A direct use of the built-in \texttt{Integrate} function does not lead to a transformation to the desired form with integrated \((e - v[x])^\frac{3}{2}\) and differentiated \(v'[x]\)...:

In[4]:= \texttt{Integrate[v'[x] v''[x] (e - v[x])^(-3/2), x]}

Out[4]= \(-3 \text{Integrate}[\frac{v'[x]^3}{\text{Sqrt}[e - v[x]] (-e + v[x])^2}, x] + \frac{v'[x]^2}{2 (e - v[x])^{3/2}}\)

Hence, we suggest the following program to carry out the integration (the rules follow from a direct calculation after changing the integration variable).

In[5]:= (* additivity *)
IntegrationOverV[f_· a_Plus * (e - v[x])^n_?(# < -1&) ] :=
  IntegrationOverV /@ Expand[f a (e - v[x])^n]

(* integration by parts *)
IntegrationOverV[a_· v'[x]^n_· (e - v[x])^n_?(# < -1&)] :=
  Expand[D[a v'[x]^n (n v - 1), x] 1/(n + 1) (e - v[x])^n (n + 1)];

IntegrationOverV[a_· v'[x] (e - v[x])^-1] := 0

(* additivity *)
IntegrationOverV[x_Plus] := IntegrationOverV /@ x

(* nothing to do *)
IntegrationOverV[x._?(FreeQ[Numerator[#, v'[x], {0, Infinity}]]&)] = x;

Now the above expression is integrated to:

In[15]:= \texttt{IntegrationOverV[v'[x] v''[x] (e - v[x])^(-3/2)]}

Out[15]= \(-\frac{2 v[x]}{\text{Sqrt}[e - v[x]]}\)

We drop the part consisting of a complete differential since it does not contribute to the integral along a closed path.

The integration by parts is more difficult to implement. The actual integration is carried out with the function \texttt{IntegrationByParts}. 
In[16]:= IntegrationByParts[x_Plus] := IntegrationByParts /@ x
IntegrationByParts[a_, (e - v[x])^n_] :=
Module[{int, intWith, intFree, intWith1},
  (* Integrate the factor a *)
  int = Integrate[a, x];
  (* analyze the usefulness of the result *)
  If[Head[int] === Plus,
    intFree = Select[int, FreeQ[#, Integrate, {0, Infinity}, Heads -> True] &];
    (* still contains undone Integrate *)
    intWith = int - intFree,
    If[FreeQ[int, Integrate, {0, Infinity}, Heads -> True],
      intFree = int; intWith = 0, intFree = 0; intWith = int]];
  (* rewrite part which contains Integrate *)
  If[intWith == 0, 
    intWith1 = If[Head[intWith] === Plus,
      Cases[intWith, Literal[_. Integrate[_, _]]],
      intWith1 = {intWith}];
    intWith2 = Plus @@ {aa_. Integrate[bb_, x] -> aa bb} /@ intWith1];
  If[intWith == 0, intWith2, 0] (e - v[x])^n +
    Expand[-intFree (-n (e - v[x])^(n - 1) v'[x]])]
IntegrationByPartsTogether carries out the integration of the collected terms with
the same denominators. myTogether collects these terms.

In[19]:= IntegrationByPartsTogether[y_Plus] :=
Module[{to, to1, ma, toInt, res},
  to = myTogether[y];
  If[Head[to] != Times,
    (* largest divergent power *)
    to1 = (Denominator /@ Apply[List, to]) /. {n_. Power[_, m_] -> m};
    ma = Max[to1];
    toInt = Cases[to, _. Power[e - v[x], pot_?(# > - ma)]];(* integrate by parts *)
    (to - (Plus @@ toInt)) + Plus @@ (IntegrationByParts /@ toInt), to]]
IntegrationByPartsTogether[x_] = x;

myTogether[x_Plus] := (* concentrate on denominators only *)
Plus @@ ((Together[Plus @@ Cases[x, _?#-1]]) & /@ Union[Flatten[Cases[#, Power[_, _], {0, Infinity}] & /@
      (Denominator /@ List @@ x)])
myTogether[x_Times] = x;

Next we give an explicit representation for the result in the form of derivatives with
respect to $E$ of definite integrals over $x$. To avoid the differentiation, we apply \texttt{HoldForm}. 
Here are the correction terms of order 1 to 8. All odd terms are complete differentials, and disappear identically. The function \texttt{WKBCorrection} applies both of the functions \texttt{IntegrationOverV} and \texttt{IntegrationByPartsTogether} “as often as possible.”
Parts of this lecture are taken from my forthcoming book "The Mathematica Guidebook", TELOS 1995, with permission of Springer-Verlag.

**Bibliography**

Simulation and Modeling Tools in Data Acquisition System Design for Future High Energy Physics Experiments

M. J. Haney  
*University of Illinois, Urbana, Illinois, USA*

**Abstract**

In light of the ever increasing complexity of high energy physics experiments, there is little choice for future experiment designers but to employ methodologies of sophistication equal or greater to those employed in developing state-of-the-art commercial electronics. This paper examines simulation and behavioral modeling, and how this approach has, and can, be applied to high energy physics experiments.

1 **Introduction**

Future high energy physics experiments face both an obvious, and an obscure challenge. With each new experiment, or upgrade, or publication, more and more of the "easy things" are being done; this obvious statement poses the challenge of ever increasing complexity in experiment design, to address ever more interesting problems. Fortunately, the electronics industry is growing rapidly in sophistication, and the elements available (ASICs, processors, data communications) are improving every day.

However, it is this same growth in sophistication in the electronics field which poses the obscure challenge. As increasingly complex building blocks are assembled, increasingly subtle system interactions and failures are becoming manifest. It is of little consolation to look at a triumph of human endeavor, such as the Hubble Space Telescope, and realize that it the optical system had only a "small" error.

Unless physics experiment designers are willing to embrace the same (or better) methodologies that are used in the development of the underlying electronics, then future experiments can anticipate only more subtle, and more debilitating results.

This paper briefly reviews simulation as it is used today in the electronics industry, and provides examples from both electronics, and high energy physics, as to what is done, and can be done in the future.

Due to the space limitations of this paper, many references and pointers have been omitted. Please refer to Section 5, Supplemental Notes, on details for access to this information.

2 **Advanced Methods**

The design of sophisticated electronics requires the use of sophisticated design automation tools. The following sections examine high level modeling, its goals and limitations, and an overview of available languages and tools.

2.1 **High Level Modeling**

Creating a model, and simulating it, represents only a small part of the design process. A brief overview of modeling is provided, and discrete event simulation in specific, followed by a discussion of synthesis, and verification and validation.
2.1.1 Modeling, Analysis, and Synthesis

In the following, the phrases "engineering modeling" and "engineering simulation" refer predominately to electrical engineering; "physics" modeling and simulation, as routinely used in data analysis, will be contrasted to this.

Modeling is the creation or fitting of a representation to a system or subsystem of interest. Models may be equations, executable code, scale mock-ups, etc. The extent to which the model approaches reality is often referred to (in engineering) as the level of abstraction. Highest level models, which reflect only the outermost visible manifestation of the system being modeled are called behavioral models. Typically, behavioral models are written in a high-level language, such as VHDL or Verilog (hardware description languages), or in C (or variants). Structural models reflect aspects of how the system is composed; an electrical schematic diagram, showing gates and registers is a structural representation. Physical models, the lowest level of abstraction, focus on the finest detail of the system construction. The masks used to produce an integrated circuit, identifying differing regions of silicon, are physical representations. These notions, behavioral, structural, and physical, tend to have a relative as opposed to absolute interpretation.

Analysis is the derivation of results from the model, typically through mathematic processes. Analyses may be closed form (analytic), or they may involve numerical methods, such as numerical integration and matrix manipulation. The circuit "simulation" program SPICE, in its DC analysis, is precisely that: analysis. Perhaps the most popular tool for performing engineering analysis is MatLab, although Mathematica is also used.

Simulation is the derivation of results by allowing the models to interact, either in time or space or whatever domain is appropriate to the problem. Simulations are "dynamic" as opposed to a static analysis.

Since the notion of "physics simulation" is familiar to anyone who has performed high energy event analysis, it is important to differentiate that which is done in physics from that which is done in engineering. Physics tools, such as GEANT, use Monte Carlo methods to create an input "stimulus" to their model (the detector), then use Monte Carlo methods to determine the outcome of local interactions (e.g. particle decay). Each input "event" is treated as a distinct, orthogonal aspect of the simulation.

Engineering simulation also may use Monte Carlo methods to generate an input stimulus, but it is more common for a specific (trace-driven) test case to be used as the stimulus. The outcome of local interactions is then determined from this stimulus in light of the system state. "Events" (in the physics sense) are not orthogonal, but in fact their interaction is the proximal cause of the system behavior.

The orthogonality issue is important. Since the outcome of each physical interaction is stochastic, it becomes exponentially difficult to keep track of all of the possibilities. Thus event-event interaction is largely ignored, which is justified as long as the occupancy of each part of the detector remains low. However, in engineering simulations, each "event" has a fairly well defined (deterministic) effect on the system state; this is the role of the model, to transform the system state. It is the evolution of this system state, in response to the input stimuli ("events" in the physics sense) which is the basis of the simulation itself.

The word "event" has a very different meaning in engineering simulation (as in "discrete event simulation"). Hence forth, "event" shall refer to any microscopic change in system state.
2.1.2 Discrete Event Simulation

There are several forms of simulation routinely used in engineering, but of special relevance to this discussion is discrete event simulation. In discrete event simulation, the simulation kernel maintains a time-sorted "event-queue", which contains pointers to models to be executed, and the time at which they are to execute. Since the event-queue is time-sorted, the kernel can simply draw the top-most "event" from the queue, set the system time equal to the time indicated for that event, and execute that model.

The model being executed then examines the system state, and based on its own functional role, transforms the system state and schedules additional events (i.e. other models) to be executed at future times. These events are "posted" into the event-queue, in time-sorted order.

The net effect of discrete event simulation is that very little overhead is incurred except with regard to those models that actually transform the system state. Thus, most of the computation of the simulation is spent in those models that do most of the work. There is, however, a non-negligible overhead associated with maintaining the event-queue in time-sorted order; this gives rise to discrete-task, and cycle based simulation methods, which are beyond the scope of this paper.

2.1.3 Synthesis

Also important to understand is the role of synthesis, which is the "automated" process of elaboration of a high level behavioral representation to (or toward) a low level structural or physical representation. Fully automated synthesis remains an elusive goal, but commercial solutions exist today for addressing the translation of RTL (register transfer level - a somewhat "lower" high level behavioral representation) to gates (structural).

2.1.4 Verification and Validation

Two final concepts are important: verification and validation. Verification, when the term is formally used, refers to the formal process of proving that the result of a synthesis is equivalent to the input specification. Mathematically, verification is an immensely difficult task. Validation is a looser term, in that it is the demonstration that a system performs as expected. Exhaustive validation could be used as verification, but the computational cost of this in any practical system is beyond prohibitive. Regrettably, the word "verification" is often abused to mean validation.

2.2 Goals and Limitations

There are several achievable goals to high level modeling and simulation. First and foremost is the validation of the system design; simulation makes it possible to determine if, and how, a system should function, prior to committing capital resources.

Simulation also supports performance estimation: throughput, deadtime, load imbalance, resource utilization. Each of these can be examined, and used as a metric for alternatives comparison.

The comparison of alternative solutions is a third significant goal of simulation. Comparisons of ATM vs barrel-shifting for event building, or push vs pull for coordinating data collection, can be readily performed through simulation. The basis of these studies can
then be used to guide system design, and assist in determining the best solution within the economic resources of the project.

However, in addition to the above lofty goals, there are quite realistic limitations to simulation. The model is no better than that which is put into it, or left out of it. Further, modeling and simulation are not inexpensive, either in CPU time, memory, or disk space. The UltraSPARC RISC developed by Sun Microsystems consumed 730 MBytes runtime memory for a full system model, and running SPICE as a test application takes a month, running at 6200 simulated instructions per second on 60 MHz Sparc20.

Simulation is also prone to propagation of error, both in the simulation itself, and in the users blind faith of its results.

2.2.1 Cases from Industry

Simulation is used very effectively in the industrial sector, making possible the complex electronic subsystems that are used as building blocks in high energy physics experiments. The following examples drawn from industry outline the benefits of simulation:

- Simulation as specification:
  The SCI specification is largely documented in C. As a result, everyone has the same information and a common interpretation. This minimizes interoperability problems.

- To get better leverage on the design process:
  Fiat Central Research reduced design cycle time by 30% by mixing schematics with VHDL; they did not want to "lose" time learning VHDL up front.

  SGI started with 78 PALs, and added 3 times more features to the specification. This resulted in 18 FPGAs (13 new) by 6 engineers, which were completed in 8 months using Verilog + Synopsys (synthesis).

  Motorola's Coldfire (embedded computing processor) was almost entirely developed in Verilog, then synthesized: 90% of the transistors were never "touched by human hands."

- To focus on the "problem" not the process:
  TransSwitch developed a SONET chip set using VHDL at the RTL level. 80% of their time was focused on functional aspects of the problem, while only 20% was spent on implementation details. As a result, major modifications had small effects on schedules. However, it became painfully clear that large scale VHDL projects are in fact large software projects; code-review and configuration management are imperatives.

  Sun Microsystems developed an UltraSPARC-I performance simulator using approximately 45,000 lines of C. Running a SPICE benchmark (15 billion instructions) takes a month, but sampling allowed a performance study to be done in 2.5 hours. In spite of a new processor, new system architecture, new silicon process, and new packaging, they were able to have multi-user UNIX running within one week of first silicon.

2.3 Languages and Tools

An every changing array of design automation tools is available; a list of commercial hardware description language vendors is available in [1].

It is relatively easy to create an event-driven simulator in an object oriented language. Defining a small number of base-classes to define an "event" with a member function of
"what_to_do", and some kernel code to support the event-queue, as well as "execute" and "post" functions provides most of the required elements. Of course, such a simple implementation places a great deal of burden on the programmer to use the pieces correctly. The real value added in creating an event driven simulator is the "ease of use" features added.

As a consequence, there are countless C/C++ based simulators available, many for free, with varying degrees of applicability. Several have been used by the high energy physics community, as well as the electronics industry. However, the electronics industry is dominated by two hardware description languages: VHDL and Verilog. Recent reviews of tools specific to these can be found in [2] and [3].

2.3.1 VHDL and Verilog

VHDL (VHSIC (Very High Speed Integrated Circuit) Hardware Description Language) was born out of a DOD mandate for documentation. IEEE Std. 1076, VHDL is a flexible language, with object-oriented aspects (such as the entity/architecture construct), supporting delayed binding (USE) and parametric instantiation (GENERATE). For various reasons, VHDL library support has not been strong, but VITAL (VHDL Initiative Toward ASIC Libraries) is addressing this deficiency, promoting "sign-off" libraries, tools, and standards.

Verilog was developed as a proprietary language by Gateway Design Systems (now owned by Cadence), yet it became the de facto standard of Synopsys, the leading synthesis tool vendor. As a result, Verilog has a large number of model libraries for ASICs. It will soon become an IEEE standard (P1364).

Perhaps the most common complaints are that VHDL has too much syntactic overhead (i.e. it takes a great deal of "boiler plate" to get anything done), while Verilog is semantically too poor (no abstract data types, no parametric instantiation, etc.) to address extremely abstract problems.

Yet Verilog remains fast to learn and easy to use. And despite persistent rumors that VHDL is sweeping the market and that Verilog is dead, there is no reason to believe that Verilog's life time will be any shorter than that of COBOL.

2.3.2 Modsim and Others

There are also many simulation-specific languages and tools to consider. The two used most often in the high energy physics are Modsim (from CACI) and Foresight (from NuThena). Modsim is a very clean object-oriented programming environment with simulation constructs built-in. It is fast to learn, easy to use, and very productive as a simulation tool. Modsim is also an excellent starting point for learning object-oriented programming fundamentals.

Foresight is a specification-focused tool. Its underlying language, Mini-Spec, is well balanced at providing both system specification, and simulation support. A graphical front-end, combined with both VHDL and C output makes Foresight a formidable tool.

3 Cases from Physics

The use of high level tools in the high energy physics community is limited but encouraging. The following cross section should provide some insight as to what is being done. There are several additional references available via the Supplemental Notes (section 5).
3.1 VHDL

VHDL has seen only a relatively small amount of use in high energy physics community, perhaps largely due to the relatively slow appearance of cost-effective tools.

A data collection chip (DCC) was modeled [4] both as a discrete device, and as a tree of DCC devices performing event collection (distributed event building), and FASTBUS simulation tools [5] were developed to provided a virtual backplane, virtual master, and virtual slave as a basis for testing new board designs in a virtual system environment.

3.2 Verilog

Verilog has been more popular than VHDL, largely due to the simplicity of the language. Unfortunately, this same simplicity has limited Verilog's usefulness in high level modeling. The language has been used to evaluate upgrade alternatives for CDF [6], and identified design improvements in the process of focusing on the model.

Verilog has been combined with DataViews (visualization and control) and Nexpert (database interface and rules driven expert system) to provide a comprehensive approach to event builder design and diagnosis [7].

And for SCI, a hardware simulation in Verilog was compared to an architectural simulation developed in Simula (to be redone in Modsim), to study latency, memory requirements, and scalability [8].

3.3 Modsim

Modsim is well suited to large scale simulations, and has been used in a number of high energy physics studies. However, a common complaint [9] is that the limits of the machine and software version have been too easy to reach.

Modsim was used in several related SDC projects; in each case the code tended not to be reused as-is, but instead was "adjusted" to suit the needs of the moment. It is not clear whether this is evidence against object-oriented code reuse, or simply a manifestation of the not-invented-here syndrome [10, 11, 12, 13].

Modsim has also seen much use at CERN. RD13 used Modsim to model a scalable DAQ system for LHC, resulting in the DAQ Simulation Library (DSL) [14]. Code is available on-line providing a wide selection of building-block objects. DAQ modeling for Atlas at CERN "distilled" DSL into SIMDAQ. Code and documentation are available on-line. Publications to appear soon (if not already). Also at CERN, RD11-EAST (embedded architectures for second-level triggering) is also using SIMDAQ.

CLEO III is using Modsim to examine its data acquisition design [15].

3.4 C++, etc.

Substantial SCI and ATM simulation has, and is, being done using C++ or variations of it [16, 17, 18]. As systems become more complex, it is increasingly important to find commercial solutions to data transmission and collection problems. But the driving forces behind ATM, for example, are not high energy physics, but telecommunications. And the problems that they are trying to address are very different from those of the physics community. It is imperative
to determine whether commercial solutions are viable, and cost effective; simulation is one relatively inexpensive way to make this evaluation.

3.5 Other

Foresight is focused as a "specification tool." It's underlying language, Mini-Spec (derived from Ada), is especially strong in abstract data types and specification. It has been under scrutiny at CERN for some time [19]. It is currently being used to model an accelerator control system at CERN. Using the FS-C code generator, a "real-time prototype" can be executed in conjunction with real control system code.

4 Summary

We are at, or perhaps beyond, the limit where one can simply buy the pieces and casually put together interesting solutions to high energy physics experiments. The consequences of assembling an expensive system that does not live up to its promises are serious indeed, not only for the group that failed, but for the high energy community at large.

To cope with exponential complexity demands approaches which transcend linearly-scaling methods. Architectural alternatives must be evaluated without making excessive financial commitments. Hence behavioral simulation. Prototyping of small parts of a problem remains valid, but system prototyping is not tractable, as system behavior typically does not scale in a simple manner. Hence virtual prototyping and mixed-level simulation.

And the time/money/thought invested in simulation must not be wasted. If the system is to be built, then a clear path from simulation to implementation must be held in sight at all times. Hence synthesis.

Finally, the only way to be confident that the system will work in the field as it does on the screen, is through an aggressive approach to testing and fault-location. Hence validation and verification.

Unfortunately, the required system-level design tools do not exist today [20]. However, the DOD sponsored RASSP (Rapid Prototyping of ASIC Signal Processors) program is a multi-company and school research project which is firmly system-oriented, and holds quite a lot of promise. Also, the marketplace is asking for increasing sophistication in cars, TV, telephone, home heating/cooling, PDA, etc. As complexity in everyday life soars, look to (market driven) embedded processor research as a driving force behind system specification and design.

5 Supplemental Notes

This paper reflects a pair of lectures provide in August/September, 1995, at the CERN School of Computing, Arles, France. The relative freedom of the lecture environment made it possible to present a somewhat wider spectrum of information related to this topic. The lecture notes, as well as a collection of reference pointers (papers, company names and telephone addresses, URLs, etc.) are available variously via the World Wide Web from

http://eng3.hep.uiuc.edu

and from the author via email

m-haney@uiuc.edu
or conventional correspondence

Michael J. Haney
High Energy Physics
University of Illinois
1110 W. Green St.
Urbana, IL 61801 USA

References

Switching techniques in data acquisition systems for future experiments.

M. F. Letheren
CERN, Geneva, Switzerland.

Abstract
An overview of the current state of development of parallel event building
techniques is given, with emphasis on future applications in the high rate experiments proposed at the
Large Hadron Collider (LHC). The paper describes the main architectural options in parallel event builders, the proposed event building architectures for LHC experiments, and the use of standard networking protocols for event building and their limitations. The main issues around the potential use of circuit switching, message switching and packet switching techniques are examined. Results from various laboratory demonstrator systems are presented.

1 Introduction

A high energy physics experiment is usually composed of several different multi-channel detectors, each of which is equipped with its own specific modular front-end and readout electronics. Whenever an event trigger occurs, readout controllers each read the data from a local group of channels and format it into an event fragment in a local front-end memory. The various event fragments belonging to an event are scattered over these frontend memories, and they must be brought together before the event can be processed on-line using algorithms operating on the global event data, or before the event data can be recorded to mass storage. The process of collecting together the distributed event fragments is called event building.

Figure 1(a) shows the architecture of a generic data acquisition system that includes a processor farm for on-line software triggering and uses a shared-medium interconnect to perform event building by moving event fragments from the sources (front-end buffers) into the destinations (members of the processor farm). All processors in the farm run identical algorithms and any given event is processed by just one processor. An event manager controls the allocation of each “new” event to a “free” processor. The shared-medium interconnect may be for example a bus (e.g. FASTBUS or VMEbus), a token-passing ring or an ethernet segment. A control protocol operates between the source and destination modules in order to provide such functions as the sequencing of the readout of event fragments, checking that all fragments are correctly received, and the detection and recovery from errors.

Before proceeding we define parameters that we will use to characterize the event builder’s performance. The rate at which events can be built must at least match the highest expected trigger rate. Related to event building rate is throughput, the quantity of event data built per unit time. The event building latency is the delay from the trigger until the last event fragment of the event is collected; it depends mainly on the time that fragments spend queuing in various parts of the system and on software overheads in the destination. The operating load is the ratio of throughput to the nominal bandwidth offered by the event builder. The operating load is a measure of the efficiency with which the hardware is used. Note that the rate is not

---

1 We will not discuss an alternative approach using the Scalable Coherent Interface [1] to implement a distributed, shared-memory, memory-mapped event building architecture minimizing data movement.
given by the reciprocal of the latency; when sufficient memory is available in the system, high rates (or equivalently, high throughput or load) can be achieved even when latencies are long (pipeline effect).

From the point of view of processing power, the architecture can be scaled to handle arbitrarily high trigger rates by adding processors. However, the shared-medium interconnect imposes a limit to the bandwidth available for moving event fragments to the destinations. Very high rate experiments require a parallel event building approach, in which events are built concurrently in the different destinations using a fabric of parallel interconnects between sources and destinations. All sources need to be able to connect to all destinations and figure 1(b) indicates how a switching network employing a crossbar architecture can be used to achieve this with one link interface per source and per destination. The switch can be electronically reconfigured to establish any desired pattern of parallel independent connections between N source and M destination modules. Until a path to the destination can be made available, each source must queue event fragments in one of M queues corresponding to the desired destination. Time multiplexing is used on the links to carry data from the different queues, while switching between different link interconnection patterns is used to route the data to the appropriate destination. A control scheme is needed to coordinate the configuration of the switch with the time multiplexing of the data on the links. In this paper we will describe switch-based event building using three different control schemes implementing message switching, circuit switching and packet switching.

The crossbar switch is said to be non-blocking because, from every source, one can always allocate a path to any destination which is not already connected to another source. The complexity of an N x N crossbar grows like N^2, and this limits the maximum practical size of a crossbar. Large switching networks use multi-stage topologies and have a complexity that grows like N.logN. Depending on the operating load and the traffic patterns, they suffer to a greater or lesser extent from internal blocking. Note that even with an internally non-blocking
switch, output blocking can occur when multiple sources try to connect simultaneously to the same destination.

There are many aspects to a complete data acquisition and triggering system, but this paper limits its scope to parallel event building architectures based on switching fabrics. We first give an overview of the main architectural options in parallel event builders and then we describe the proposed event building architectures for LHC experiments, with emphasis on performance and other requirements. We then look at the use of standard networking protocols for event building and their limitations. This is followed by an overview of the issues around the potential use of circuit switching technologies for event building. Work on the alternative approach of using message switching technologies is then described and is followed by considering the application of packet switching and cell switching techniques.

2 Architecture Options for Parallel Event Building

2.1 Push versus Pull control protocols

The data flow control architecture can use either the push or pull discipline. In the push discipline, the event manager assigns a destination for the next event and broadcasts the event number and destination identifier to all sources. The sources then send out their event fragments to the assigned destination. This requires a minimal protocol overhead and has potentially the highest throughput. However, because multiple sources attempt to send their event fragments concurrently, the event fragments will arrive at the destination in an indeterminate order, requiring a scatter-gather hardware feature in the interface adapter, or a more intensive activity for buffer management and merging in the host.

One or more sources may be either “dead” or “empty” for a given event, and therefore, in the push architecture, the destination must implement some algorithm (as part of the event building protocol) that allows it to decide when all the event fragments for an event have been received. In addition, multiple sources compete for the same output and, depending on the switching technology and the algorithm used to assign events to destinations, the result may be reduced throughput, increased event building latency, or loss of data. In section 6.2.3 we show that these effects can be minimized by an appropriate destination assignment algorithm, and in section 7.2.2 we show how the traffic shaping technique can resolve these problems.

In the pull discipline the destination processor initiates the data transfer by requesting event fragments from each of the sources in turn. The event fragments are therefore delivered in a known, fixed sequence, and it is implicitly clear when all the event fragments have been collected. In addition, error detection and handling are relatively straightforward. The pull discipline can be used to implement intelligent, selective readout, thereby reducing the amount of data moved and allowing the use of smaller and cheaper switches. The sequential pull of fragments, or multicasting of requests to small groups of sources, avoids the congestion and blocking problems of the push architecture. The disadvantage is that the pull technique imposes a fair amount of software overhead to support the source-destination control protocols.

2.2 Event flow management algorithms

The event manager’s choice of destination for the next event may use strategies such as round-robin, random destination, least loaded processor, etc. Ideally the event manager should balance the load on the members of the processor farm, but, as already mentioned, the choice of the destination assignment algorithm can have a significant impact on the performance of the switching fabric.
2.3 Error detection and recovery

The control protocols for switch-based event building will be layered on the physical and link layer protocol layers of the chosen link and switching technologies. These lower layers will include error detection and perhaps error correction capability. The event building protocol layers will have to handle errors passed to them from the lower layers, deciding for example how to handle a corrupted or lost event fragment. In addition they will have to monitor the validity of event fragments, and signal problems (e.g. dead sources, sources sending corrupt data, etc.) to a higher layer.

2.4 Phased event building

The generic architectures described in section 1 have been over simplified in order to introduce the basic concepts. In practice high rate experiments use multiple levels of online software triggering and may use phased event building schemes in order to reduce the required bandwidth for data movement. In phased event building, initially only a part of the event’s total data are moved into the processor farm. A rapid decision is made based on this subset of the event data. For the small fraction of events that are accepted after the first phase, a second phase of event building is started in order to collect additional event data, on which more sophisticated analysis can be performed to further refine the selection of events. Multiple phases of event building can continue until the full event data is accessible to the on-line software trigger.

2.5 Switching architecture

As previously mentioned, each source (and destination) has one physical link to the switch fabric which is shared by the different logical connections to the destinations (sources). We will consider parallel event builders employing three different switching techniques distinguished by the way in which the logical connections share the physical links. Link sharing can be by synchronous or asynchronous time division multiplexing (TDM), or by sequentially opening and closing dedicated paths to the desired destination(s). Table I compares the three switching architectures, which will now be described in more detail.

2.5.1 Synchronous transfer mode and circuit switching

When synchronous TDM is used the data are said to be transported in the Synchronous Transfer Mode (STM). In telecommunications networks, in which developments were dominated by the requirements of voice communication, multiple channels are time multiplexed on a link using fixed-length time slots. On a regular cycle, each subscriber-to-subscriber connection (called a circuit) is allocated a time slot, during which it can transmit a voice sample. A global timing reference is provided by grouping time slots into a frame that repeats every 125 µs, and each circuit is allocated one time slot in a fixed position within the frame. When voice samples are switched in the exchange, the circuit, or equivalently the destination, of the sample is inferred from its position within the time frame.

---

2 We use the terminology of the ISO Open Systems Interconnection (OSI) reference model [2].

3 In some standards, errors may be detected at these layers, whereas recovery may have to be implemented in the higher layers (if errors are infrequent, correcting them at a higher layer is simpler and leads to better overall system performance).
As an example, figure 2 shows the frame structure specified in the International Telecommunication Union’s (ITU) recommendation G.703 [3] for the transmission of circuits at the bit-rate of 34.368 Mbit/s. The frame is presented as a matrix of 59 columns (each of one byte) and 9 rows. One byte of the matrix is reserved and, together with an additional 6 bytes, forms the so-called **path overhead**, which is used for link error signalling and “operations and management” (OAM) functions. The total length of the frame is 537 bytes and it is transmitted in exactly 125 µs at the bit-rate of 34.368 Mbit/s. When a time slot of length eight bits is used, the available 530 byte payload can carry 530 circuits of 64 kbit/s each.

<table>
<thead>
<tr>
<th>Multiplexing Scheme</th>
<th>Switching Technique</th>
<th>Application Area and Characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Synchronous TDM (STM)</td>
<td>Circuit switching</td>
<td>Telephone switching technology; constant bit-rate traffic; equal bandwidth per circuit; concurrently active circuits.</td>
</tr>
<tr>
<td>Asynchronous TDM (ATM)</td>
<td>Packet switching</td>
<td>Data network switching technology; bursty traffic; bandwidth allocated per connection; concurrently active connections.</td>
</tr>
<tr>
<td>Dedicated connection</td>
<td>Message switching</td>
<td>Switching streams point-to-point; connection setup overheads; efficient for long block transfers; sequentially active connections.</td>
</tr>
</tbody>
</table>

**Table I**: Traffic multiplexing schemes and their associated switching techniques.

As an example, figure 2 shows the frame structure specified in the International Telecommunication Union’s (ITU) recommendation G.703 [3] for the transmission of circuits at the bit-rate of 34.368 Mbit/s. The frame is presented as a matrix of 59 columns (each of one byte) and 9 rows. One byte of the matrix is reserved and, together with an additional 6 bytes, forms the so-called *path overhead*, which is used for link error signalling and “operations and management” (OAM) functions. The total length of the frame is 537 bytes and it is transmitted in exactly 125 µs at the bit-rate of 34.368 Mbit/s. When a time slot of length eight bits is used, the available 530 byte payload can carry 530 circuits of 64 kbit/s each.

![Figure 2: The 125 µs G.703 frame structure used for synchronous transmission at 34.368 Mbit/s.](image)

The STM data are switched in the exchange using the so-called *circuit switching* technique, whose principle is indicated in figure 3. The switch first synchronizes frames on incoming links, and then maps slots from the frames on each incoming link into new slots in frames on an outgoing link. In general, in the public switching networks the time slots used to carry a circuit on the incoming and outgoing links are not in the same fixed position within their respective frames. Therefore the STM switch will be built from a combination of time slot interchange (TSI) units and space switches, which together perform the reordering of slots in
the time frames and the routing of slot data from input link to output link. The TSI is essentially a memory in which the time slots are organised into separate logical queues, that can be randomly accessed by the switch. The TSI and switch routing control information are shown in tabular form in the figure. This table is set up during the signalling protocol that establishes the circuits between the subscribers (the call dialling process).

![Figure 3: The principle of circuit switching. Time slots are switched between input and output time slot interchangers (TSIs) under the control of a mapping table that, for each time slot in the frame of each input link, defines (i) the switch configuration needed to route the time slot to the appropriate output link and (2) the new time slot position in the frame on the outgoing link.]

<table>
<thead>
<tr>
<th>Circuit</th>
<th>In-link slot #</th>
<th>Out-link slot #</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>c</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Circuit</th>
<th>In-link slot #</th>
<th>Out-link slot #</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>e</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

An important characteristic of the circuit switched technique is that it offers equal bandwidth to all circuits. Circuit switching is not well suited for supporting a mixture of different classes of services, such as computer network traffic, telephone traffic, compressed video, or high definition television (HDTV), each of which requires circuits with widely different bandwidths (from 64 kbit/s to 140 Mbit/s). In addition computer network and compressed video traffic are naturally bursty; circuit bandwidth is wasted when the sender is silent or not transmitting at the peak bandwidth allocated to the circuit. Such applications are better handled by:

2.5.2 Asynchronous transfer mode and packet switching

When the time division multiplexing of the link uses an asynchronous discipline, the data are said to be transported in the Asynchronous Transfer Mode (ATM).

Here the acronym ATM is used in a generic sense, and is not to be confused with the specific ATM technology standardized for B-ISDN and LAN switching, which is introduced later in this section.
The switching technique associated with asynchronous TDM is known as packet switching, and it uses the packet’s label (or destination address) to decide to which output link the packet should be routed, as shown in figure 4. The label can be thought of as identifying a virtual connection on which all packets with the same label travel through the system. Because subscribers can send variable length packets and can seize link bandwidth on demand, this mode offers a more flexible service for intermixing virtual connections that require different bandwidths. Because bandwidth is only used when packets are actually transmitted it can also efficiently carry bursty traffic (where the instantaneous bandwidth utilization of a virtual connection fluctuates in time). Bandwidth which is not used by a virtual connection is available for use by other virtual connections. The resources of link and switch are shared between virtual connections by statistical multiplexing. For a large number of independent connections the statistical fluctuations of individual connections average out, and the switch can run at a high average load.

The packet switching hardware uses the packet label or destination address to decide to which output port it shall send the packet. In ATM the correspondence between the label value, the output port number and the new label value of the cell on the output link is contained in tables in the switch hardware. As for the circuit switched case, these tables are initialized by a signalling protocol which establishes the virtual connections.

The flexibility of packet switching lead to its use in computer networking technologies such as ethernet, the Fibre Distributed Data Interface (FDDI), etc. More recently, this flexibility to intermix and efficiently support the differing traffic characteristics and quality of service requirements of applications, such as video and audio distribution, video conferencing, and computer data transport, has lead to it being chosen as the underlying mechanism for the Broadband Integrated Services Digital Networks (B-ISDN). The telecommunications industry has standardized a particular asynchronous transfer mode technology for the B-ISDN [4], which is based on packets with a fixed length of 53 bytes. The particular asynchronous transfer mode technology used by the B-ISDN has now subsumed the use of the acronym ATM, and from here onwards we will implicitly refer to the B-ISDN version of the asynchronous transfer mode.
whenever we use the acronym ATM. The 53-byte packets used for the ATM transmission and switching are called cells in order to differentiate them from the generic use of the term “packet” or the specific packets used at higher levels of data communications protocols. The particular technology of ATM switching is called cell switching.

The basic ATM standard defined by the ITU is being adopted by the computer industry for high speed networking, and the ATM Forum [5] is further developing the standards with emphasis on the computer industry’s requirements.

The Synchronous Digital Hierarchy (SDH) [6] and the Synchronous Optical Network (SONET) [7] are (almost) identical framing standards that have been designed to support the multiplexing of all previously defined synchronous transmission standards in a common STM frame. However, SDH and SONET also support the asynchronous TDM transport of ATM cells by loading them into the first available position within the frame payload. The SONET / SDH standards are widely used for transport of ATM cells at the physical layer within both the telecommunications and computer industry. In addition, the ATM Forum has defined several alternative physical layer transport standards for use with ATM.

2.5.3 Dedicated point-to-point links and message switching

In contrast to the circuit switching and packet switching techniques, which use time division multiplexing to concurrently carry the traffic of multiple circuits (connections) over a link, the message switching approach sets up a dedicated path between source and destination for the time required to transmit a complete message (in our case an event fragment). Once the message has been transmitted the path is torn down and a new dedicated path can be set up to the next destination.

The delay required to set up a connection imposes a deadtime, during which no data are transferred. As shown in figure 5, this overhead consists of one component contributed by the routing of a connection request through the switch, and a second component contributed by the delay before the receiver signals back a connection accept. The effective throughput that can be achieved depends on the ratio of the connection set up overhead to the time required to transmit the message.

![Figure 5: Message switching involves an overhead in order to set up a dedicated connection.](attachment:image)

Examples of message switching technologies, originally developed for high speed channel connections between processors and/or peripherals, are the High Performance Parallel Interface (HiPPI) standard [8] and the Fibre Channel [9] standard (in service class 1).
2.5.4 Signalling

In telephone or typical computer networks, a *signalling protocol* must be used to establish a circuit or (virtual or dedicated) connection between source and destination before data can be transferred. Signalling sets up a contract between the two connection end points (source and destination) and the switching fabric. On the other hand, in event building the full connectivity between all sources and destinations is used, and no connection is dormant for long periods. In the circuit switched or packet switched event builder we can take advantage of this fact by defining circuits or virtual connections which are left in place throughout the data taking run. They can be initially established either by running signalling protocols, or by adopting a system-wide convention that allows off-line calculation and subsequent downloading of the mapping tables for the switches. By using such semi-permanent circuits or virtual connections we avoid overheads associated with dynamic connection set up.

By contrast, in the case of event building with message switching, the use of dedicated connections forces a signalling protocol to be executed for every message transferred, and results in a connection set up overhead as indicated in figure 5.

3 Proposed Architectures and Requirements for the LHC Experiments

We describe next the phased event building schemes proposed for the CMS and ATLAS experiments at the LHC, and use these to define event builder requirements.

3.1 Phased event building in CMS

The CMS collaboration proposes to use a single large switch and processor farm [10] to perform event building and filtering in two phases, as shown in figure 6. The first phase (which they call “virtual level-2”) operates at the full level-1 trigger rate (100 kHz) and builds the event data from a subset of the detectors (the calorimeter, preshower and muon detectors). The event fragments are stored in front-end dual port memories (DPMs) and the events are built in similar dual port memories (Switch Farm Interface - SFI) from where they are passed to a processor which then executes the second level trigger algorithm. All processors of the farm are used for

![Figure 6](image-url)

*Figure 6:* The phased event building scheme proposed for the CMS experiment.
the second level trigger. Whenever one of the processors finds an event that passes the second level trigger, it initiates the second phase of event building, in which the remaining data for that event (from the tracker) are assembled into the SFI. Once the entire event data is present the processor proceeds to execute the level-3 trigger algorithms.

Note that the DPMs have to emit event fragments at the full level-1 trigger rate of 100 kHz. The protocol with the event manager, the access of the appropriate event fragment queues, the adaptation of the data to the switch interface and the memory management of the DPM is a formidable challenge at these rates. Considerable effort is being expended on the development of programmable and hardwired DPM controllers [11].

### 3.2 Phased event building in ATLAS

Figure 7 shows the logical model of the ATLAS architecture [12] and the proposed event building phases. The level-1 trigger provides the level-2 trigger system with pointers to regions of the detector containing information that fired the level-1 trigger. The “regions of interest” (RoI) pointers change from event to event; the RoIs are dynamically selected for readout by the level-2 trigger system; the data of each RoI is processed in a different local processor (LP) in order to extract compact descriptions (<100 bytes) of the physics features (e.g. a shower in the calorimeter). After the local processing step, each event’s features are sent to one of a farm of global processors (GP), where a level-2 trigger decision is made. The level-2 decision may not use the information from all detectors. Thus the level-2 system uses partial event building in local and global phases over the local and global networks shown in figure 7. For those events that pass the level-2 trigger, there follows a full event building phase for the level-3 trigger system.

![Figure 7: The phased event building scheme proposed for the ATLAS experiment](image)

The model shown in Figure 7 is a logical model; it could be implemented in many ways, for example by using physically distinct networks and processor farms for each phase, or by handling the three logical phases with a single farm and a single large flexible switching system.
The average number of RoIs per event is expected to be about 5, and their selective readout is expected to reduce the effective level-1 trigger rate (T1) to be handled by individual front-end buffers to approximately 10 kHz (c.f. 100 kHz in CMS), which gives more scope for development of an intelligent front-end buffer [13]. In principle, the parallel processing of RoI’s also reduces level-2 data collection and trigger processing latency with respect to the equivalent latencies in the CMS architecture. However, given the availability of large, affordable frontend buffer memories, this does not seem to the author to be a decisive advantage.

3.3 Requirements for event building at LHC

3.3.1 Bandwidth and latency requirements

The data acquisition systems of the LHC experiments [10, 12] are being designed to handle a maximum first level trigger rate T1 ~ 100 kHz, corresponding to operation at the nominal full LHC luminosity of $10^{34}$ cm$^{-2}$s$^{-1}$. In order to estimate the bandwidth requirements for the phased event building system, we reproduce in table II the expected event data sizes for each subdetector of the CMS experiment. The tracker and pixel detector together produce the largest contribution to event data, which is why they are not used in the virtual level-2 phase. It is (conservatively?) estimated that the virtual level-2 phase will reduce the level-1 rate T1 ~ 100 kHz to a level-2 accept rate T2 ~ 10 kHz.

Table II also shows the estimated maximum event building traffic to be handled by the CMS virtual level-2 and level-3 phases under the above assumptions. The event builder should support an aggregate throughput of ~25 GByte/s, or approximately 200 Gbit/s. In practice it is expected to be difficult to operate a large switch above 50% of its nominal aggregate bandwidth, therefore CMS have specified a total switching bandwidth of ~ 500 Gbit/s. A suitable CMS event builder could be implemented for example with a large switching fabric having 1000 input and 1000 output ports, each running at the nominal standard SONET bit-rate of 622 Mbit/s.

<table>
<thead>
<tr>
<th>Sub-detector</th>
<th>No. channels</th>
<th>Occupancy (%)</th>
<th>Event size (kByte)</th>
<th>Event builder bandwidth (GByte/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>L2 traffic (T1=100 kHz)</td>
</tr>
<tr>
<td>Pixel</td>
<td>80 000 000</td>
<td>0.01</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>Inner Tracker</td>
<td>16 000 000</td>
<td>3</td>
<td>700</td>
<td>-</td>
</tr>
<tr>
<td>Preshower</td>
<td>512 000</td>
<td>10</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>Calorimeters</td>
<td>250 000</td>
<td>10</td>
<td>50</td>
<td>5</td>
</tr>
<tr>
<td>Muons</td>
<td>1 000 000</td>
<td>0.1</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>L1 trigger</td>
<td>10 000</td>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>TOTALS:</td>
<td>~ 10^8</td>
<td>-</td>
<td>~1000</td>
<td>17</td>
</tr>
</tbody>
</table>

*Table II:* Average sub-detector event size and expected event building bandwidth contributions for the CMS experiment (drawn from [10]).

The required aggregate event building bandwidth for ATLAS is somewhat smaller than for CMS, due to the selective readout of RoIs, and is estimated to be of the order 100 Gbit/s.

In the CMS architecture, each of the 1000 destinations must make on average 100 level-2 decisions per second, i.e. one every 10 ms. If the processing of events is overlapped with event building, the event builder must deliver a new event at each destination every 10 ms on average. The latency for building an event can be longer than 10 ms if several events are concurrently
built in each destination. The upper limit of acceptable event building latency will be determined by the number of concurrently built events that can be supported by the switch interface hardware and software, and not by the available buffer memory (the planned 100 MByte capacity of the CMS DPMs is sufficient to buffer the full level-1 data stream for 1 second). A few times 10 ms is an acceptable latency for building a single event for level-2. For level-3, acceptable event building latencies are up to a few hundred ms.

3.3.2 Other requirements

There are several important “soft” requirements to be taken into consideration. The event builder must be expandible, and exhibit favourable scaling characteristics for throughput and latency as a function of offered load. The chosen technology should follow an open communications standard that will ensure plug-compatibility with equipment from different manufacturers, and the long term commercial availability of system components (interface adaptors, protocol chip sets, etc.). Ideally, the life cycle of the switch itself should match that of the LHC experiment.

In view of the scale of the LHC event builders, a fault tolerant switch architecture would be desirable, and good operations and management tools for monitoring of performance and error rates, testing and fault diagnosis will be necessary. The switching fabric should support partitioning into individual private networks for the independent test and development of detectors by their dedicated teams, and it should provide the ability to spy on the data streams.

A rich range of switching technologies is currently under study as potential candidates for constructing high rate event builders [e.g. 1, 14-17]. In this paper we select a few standardized, commercially supported technologies to illustrate the pros and cons of the three different switching architectures described in section 2.5.

4 Event Building using the TCP/IP Protocols

A number of experiments [18-20] are using farms of high performance UNIX workstations to perform on-line event filtering with (a possibly reduced version of) the off-line event analysis software. Front-end readout processors running a real time operating system assemble the event fragments. The events are built in the UNIX workstations using a high performance network and commercial network adaptors. The standard internet communications protocols TCP/IP, which are supported in both the UNIX operating system and the real time kernel of the frontend processor, are used to transmit event fragments. This approach minimizes the amount of special hardware and protocol software development needed.

4.1 Overview of TCP/IP

Figure 8 shows the TCP/IP protocols in terms of the seven layer OSI protocol reference model [2]. The OSI physical and link layers implement the underlying network technology-dependent functions that handle the transmission of data in blocks or packets. The Internet Protocol (IP) corresponds to the OSI network layer; it hides the underlying technology from the higher levels and guarantees interworking between networks based on different technologies. The IP layer forwards data in IP packets (length up to 64kByte) and fragments the IP packet into the technology-dependent block or packet formats used at the link layer. IP provides a connectionless service, i.e. it does not open a connection with the destination when sending the packet, and it provides no end-to-end reliability functions or flow control.
The Transmission Control Protocol (TCP) is layered on IP and corresponds to the OSI transport layer. It provides a connection-oriented service by supporting virtual connections between a source application and a destination application. The connection must be set up before messages can be exchanged. TCP ensures reliable, in-order delivery with error detection and packet re-transmission after acknowledgment timeouts. A sliding window data flow scheme increases throughput on long connections by allowing the sender to transmit ahead without waiting for acknowledgement.

**4.2 Event building with TCP/IP**

The UNIX operating system allows interprocess communication over TCP/IP via a de facto standard set of system calls known as the sockets interface. The sockets interface can be used to transfer event fragments between the front-end processors (running a UNIX-like real time operating system with support for TCP/IP sockets) and the UNIX host in which the events are to be built and filtered.

The use of TCP/IP for event building appears attractive because it ensures manufacturer and technology independence, as well as providing “for free” many important features (error checking, re-transmission etc.) that would otherwise have to be developed by the DAQ systems designer. However, TCP/IP implementations may not be optimally layered on any given technology, and the fact that they are accessed via operating system calls means that performance is probably limited by operating system overheads and the copying of data buffers between user-space and kernel-space. An additional penalty in using TCP/IP for event building is that the IP layer is largely redundant because its primary function (providing interworking in an inhomogeneous network) is not required. In addition, although TCP provides flow control on each connection, it cannot provide the flow control that is needed to manage congestion when independent TCP/IP connections concurrently send traffic to the same output port. Such congestion must be resolved by the network hardware or minimized by the system designer applying a traffic shaping technique, as we will describe in section 7.2.2.
4.2.1 TCP/IP performance measurements

Measurements have been made to understand whether TCP/IP-based event building could be applied to high rate data acquisition systems using packet switched network technologies such as Fibre Channel or the ATM. Table III summarizes the measurements made in [21] using a 266 Mbit/s Ancor CXT 250 Fibre Channel switch, IBM RS/6000-250 (66 MHz, 63 integer SPEC) and RS/6000-590 (66.6 MHz, 122 integer SPEC) UNIX workstations equipped with Fibre Channel adaptors from Ancor.

<table>
<thead>
<tr>
<th>Transfer</th>
<th>TCP/IP Overhead</th>
<th>Message Latency</th>
<th>TCP/IP throughput (64 kByte message)</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS6000-250 -&gt; switch -&gt; RS6000-590</td>
<td>610 µs</td>
<td>1104 µs</td>
<td>10.3 MByte/s</td>
</tr>
<tr>
<td>RS6000-590 -&gt; switch -&gt; RS6000-250</td>
<td>365 µs</td>
<td>1092 µs</td>
<td>12.1 MByte/s</td>
</tr>
<tr>
<td>RS6000-250 -&gt; RS6000-590</td>
<td>610 µs</td>
<td>1025 µs</td>
<td>10.3 MByte/s</td>
</tr>
<tr>
<td>RS6000-590 -&gt; RS6000-250</td>
<td>365 µs</td>
<td>1025 µs</td>
<td>12.1 MByte/s</td>
</tr>
</tbody>
</table>

Table III: Measured overheads, communication latencies and throughputs using TCP/IP to pass messages between UNIX workstations over a 266 Mbit/s Fibre Channel switch [21].

In the table, *overhead* is defined as the time from an application sending a message to the time when it is ready to send a new message, not counting the time to transfer data; it was evaluated by measuring the time to perform the write of a one byte message. The reciprocal of the overhead defines an upper limit to the rate at which (zero length) event fragments can be pushed out. The *latency* was measured as half of the round trip delay for bouncing a 1 byte message back to the sender, and *throughput* was measured as the volume of data transmitted per second when using a large message of 64 kBytes.

The TCP/IP overhead was found to be dominated by software and to scale approximately with the power of the CPU used. The observed overhead limits the maximum rate of sending event fragments to a few kHz. The measured best TCP/IP throughput was limited to 12 MByte/s by available CPU power (the observed CPU utilization was of the order 90%). The observed throughput increased to 16 MByte/s when the TCP/IP protocol layers were bypassed by using a light weight “direct channel” protocol supplied by Ancor. The switch connection set up latency (hardware) only contributes 70 or 80 µs to the total latency.

Measurements of transfers made between two Digital Equipment Corporation Alpha workstations with TCP/IP running over a Digital Equipment Corporation Gigaswitch using 155 Mbit/s ATM link adaptors achieved a throughput of approximately 130 Mbit/s for large messages [22]. However, throughput dropped significantly for message lengths below 5kByte. For messages of 1 kByte, throughput was of the order 40 Mbit/s.

These measurements demonstrate that the software overheads associated with TCP/IP make it unsuitable for applications where short communications latencies or high throughput are required, or where short messages need to be sent at high frequencies. The impact of the TCP/IP overheads can be minimized by packing multiple events into each message (see for example [19]). Even so, when using 1 Gbit/s technology, current high performance workstations can only sustain throughputs corresponding to approximately 10% of the link bandwidth. Clearly this situation will improve as more powerful processors become available and/or network adaptors incorporate dedicated protocol processors.
In summary, the poor performance of TCP/IP with small packet lengths and the CPU-limited throughput obtained with larger packets make it unsuitable for high rate experiments such as ATLAS and CMS. However, TCP/IP could find application in the ALICE [23] heavy ion collider experiment, where the event rate is expected to be ~50 Hz and the event size around 20 MByte. E.g. by choosing a 128 x 128 Fibre Channel switch with link speeds of 266 Mbit/s, and running TCP/IP on ~100 MIPS processors, we would perform event building with average fragment sizes of ~200 kByte. Under these conditions TCP/IP software overheads would be insignificant and one should be able to achieve approximately 40% load on the switch. In pp-collider operation the ALICE event fragments would be of size ~200 Byte and the event rate around 500 Hz, which is low enough that TCP/IP overheads would not limit throughput.

High performance event building with single events requires the elimination of operating system overheads and the layering of event building protocols directly on the network layer.

5 Parallel Event Building using Circuit Switching

The adaptation of the telephone circuit switching technique described in section 2.5.1 to parallel event building was first reported in [24]. An N x N non-blocking crossbar switch was used, and the input and output time slot interchangers shown in figure 3 were implemented by dual ported memories (DPM). As indicated in figure 9(a), each input DPM contained a logical queue for each destination, into which were placed the event fragments to be sent to that destination. Each destination DPM contained a logical queue per source, in which the event fragments coming from that source were assembled. A global control scheme defines the time slots and synchronizes the configuration of the switch with the enabling of the appropriate queues in the source, and destination modules. Figure 9(b) indicates how the controller globally orchestrates the set up and tear down of connections in successive time slots, so that no two sources simultaneously connect to the same destination, and all sources regularly connect to all destinations in a cycle (corresponding to the length of the time slot frame). Figure 9(c) shows how, using a round-robin destination assignment algorithm, successive events can be built in successive output DPMs.

This simple barrel shifter scheme avoids output blocking and is very efficient when all event fragments have equal size and the time slot corresponds to the transmission time of one event fragment; in this case one complete event finishes per time slot. The throughput of the event builder scales linearly with the size of the crossbar switch. However, when event fragment sizes are variable, bandwidth efficiency drops because time slots are only partially filled.

AT KEK, a custom built barrel shifter event builder is being developed for possible use in a B-physics experiment [25]. High bandwidth efficiency is maintained by performing concurrent building of multiple events in each destination. As indicated in figure 9(d), when a source has finished transmitting the current event fragment, it fills the remainder of the time slot by starting transmission of the next event fragment in the currently selected queue. The optimal choice of time slot size [26] was found to be equal to the average event fragment size (larger time slots increased the required queue lengths in source and destination, while smaller time slots increased the impact of overheads associated with switching between time slot states).

6 Parallel Event Builders based on Message Switching

6.1 Event building with transputers

A good example of the message switching approach is provided by event building for the
The Zeus event builder is constructed from T800 transputer-based modules and uses a custom made 64 x 64 crossbar switch to build events from 17 sources into 6 destinations. Each source has 3 links to the switch and each destination has 8 links to the switch. Links are based on the T800 transputer's serial link (nominal throughput 20 Mbit/s) and the switch is built from four C004 32 x 32 crossbar switch chips. Source and destination modules were constructed using T800 transputers and triple-ported memory to hold the event fragments and the built events. The software in the transputers runs in stand-alone mode (no operating system).

A router and controller module, based on a transputer, runs software that allocates events to the least busy destinations. As soon as an event fragment becomes available in the 3port memory of a source, the controller module is requested to set up a connection to the allocated destination via the crossbar. The event fragment is transferred using the T800 link protocol and then the connection is released. Since sources make their requests asynchronously, output blocking can occur if a source tries to connect to a destination that is already connected to another source. In that case the connection request is queued until the output becomes available. The T800 link throughput is limited to 600 kByte/s by the delay of the handshaking protocol operating over the 90 m of cable between the source and destination modules. The event builder

**Figure 9:** Event building with a simple synchronous circuit switched barrel shifter; (a) source and destination queues; (b) switch configurations in successive time slots; (c) fragment sequencing on the links for constant sized fragments, one event built per destination; (d) fragment sequencing for variable sized fragments, multiple events built per destination.

level-3 trigger farm in the Zeus experiment [27]. The Zeus event builder is constructed from T800 transputer-based modules and uses a custom made 64 x 64 crossbar switch to build events from 17 sources into 6 destinations. Each source has 3 links to the switch and each destination has 8 links to the switch. Links are based on the T800 transputer’s serial link (nominal throughput 20 Mbit/s) and the switch is built from four C004 32 x 32 crossbar switch chips. Source and destination modules were constructed using T800 transputers and triple-ported memory to hold the event fragments and the built events. The software in the transputers runs in stand-alone mode (no operating system).
can sustain an aggregate bandwidth of 24 MByte/s, which is sufficient to meet the Zeus design goal of handling at least 100 events/s of average size 150 kByte.

The transputer architecture offers an efficient integration of distributed multiprocessing and interprocess communication (the most recent transputer technology uses packet switching) that makes it particularly suitable for building homogeneous, scalable data acquisition systems [28, 29]. The construction and evaluation, for event building applications, of a large high performance packet switch based on OMI/HIC link and switching technology [30] is being carried out at CERN [17].

6.2 Parallel message switched event building with HiPPI and Fibre Channel

A similar approach to that adopted for the Zeus event builder could be taken with commercial HiPPI or Fibre Channel (class 1) switches. Both of these technologies use message switching by including, at the head of the bit stream sent over the link, information which is intercepted by the switch and used to set up dedicated, full bandwidth connections5.

6.2.1 HiPPI technology

HiPPI was originally developed as a point-to-point simplex (unidirectional) connection technology for high speed data transfer between mainframes and peripherals using a 32-bit wide data path (copper cable) and a 25 MHz clock, resulting in a bandwidth of 800 Mbit/s. HiPPI sender and receiver use a simple connection set up protocol based on a connection REQUEST line, asserted by the sender, and a CONNECT acknowledge line, which is asserted by the receiver if it accepts the connection request. Once the connection is established, one or more variable length packets (the HiPPI packet is not a packet in the sense of packet switching because it contains no label or address) can be sent. The HiPPI packets are broken down into bursts of 1024 bytes. A credit-based flow control mechanism is supported in which the receiver gives a number of “credits’’ to the sender by pulsing a READY line (one credit per pulse). The sender may send one burst per credit until it has exhausted its supply of credits. The connection is broken either by the sender dropping its REQUEST line or the receiver dropping its CONNECT line. Byte parity is used on the datapath and a checkword follows each burst.

HiPPI has been extended to support message switching by augmenting the connection set up protocol with the transmission of a 4-byte I-field. The I-field is used by a switch to establish a path to the receiver, which can then issue CONNECT acknowledge. If the connection request cannot be routed to the destination, either because the switch’s output port is already occupied or because of internal blocking in the switch, the sender can wait for the connection to be established, or it can drop the request (and possibly try to connect to a different receiver). HiPPI is a very simple protocol which suffers very little throughput overhead caused by header/trailer information. Typical connection set up overheads are ~ 1 µs and have little impact on effective throughput, even for short messages and fast switching rates.

---

5 Often HiPPI and Fibre Channel (class 1) are referred to as circuit switched technologies because, during the lifetime of a connection, an unbroken chain of dedicated links is reserved between source and destination to transmit, at full speed, the data associated with the connection; there is no multiplexing on the links of data from other connections during the lifetime of the connection. This technique is equivalent to the early dedicated circuit switching performed in telephone exchanges. In this paper we prefer to use the term message switching in order to draw a distinction with the synchronous TDM and switching techniques used in modern telephone exchanges (as previously described in section 2.5.1).
### 6.2.2 Fibre Channel technology

The Fibre Channel protocol, as its name implies, was originally developed as a high performance full-duplex switched interconnect technology for communicating large blocks of data at high speed between processors and peripherals. However, the standard was later extended to support general purpose packet switched networking. The Fibre Channel standard [9] defines five protocol layers (FC-0 through FC-4).

The FC-0 layer defines the physical media, connectors, and standard bit-rates to be used on Fibre Channel links, which always consist of a pair of fibres/wires (one in each direction). The standard bit-rates are 132.8, 265.6, 531.25 and 1062.5 Mbit/s, and are supported by media ranging from shielded twisted pairs, through coax, multimode and monomode optical fibre. Fibre Channel switches available today operate at 266 Mbit/s; 1 Gbit/s switches are under development.

The FC-1 layer defines clock and data encoding in the serial bit stream, with detection of transmission errors. The method used is called 8B/10B because 8-bits of parallel data are converted into 10 bits in the serial stream.

![Diagram](image)

**Figure 10:** Effective maximum obtainable user-data throughput over a class 1 switched Fibre Channel link as a function of the length of the user data message and the connection set up and tear down overhead ($T_c$). Throughput drops for short messages due to the fixed 36-byte header/trailer overhead of each frame. No overheads are included for software or protocol chip set operation, except for the case of the TCP/IP curve, where measurements reported in section 4.2.1 showed that software imposes $T_c \approx 365 \mu s$ and limits maximum throughput to 105.9 Mbit/s for long messages.

The FC-2 layer defines the transport mechanism between nodes (N-ports) communicating either via a point-to-point Fibre Channel link, or a switch. The basic protocol data unit is the Frame, which can carry a maximum user data payload of 2112 bytes and has a header/trailer overhead of 36 bytes. FC-2 uses a look ahead flow control mechanism where each frame is acknowledged by the receiver. At the FC-2 layer, Fibre Channel defines three basic service classes:

- **Class 1** supports message switching via dedicated, full bandwidth N-port to N-port connections. An end-to-end dedicated path has to be reserved before data can be transferred. As shown in figure 10, depending on the length of the user data,

---

6 The FC-2 frame is not to be confused with the previous use of the term frame in section 2.5.1 in the context of the synchronous transport mode.
connection set up and tear down overhead can significantly impact effective throughput. Class 1 is optimized for the fast transport of large blocks of data.

- **Class 2** and **Class 3** support the packet switching approach by providing a connectionless frame switched service. The main difference between classes 2 and 3 is that class 2 guarantees delivery and provides acknowledgment of delivery (or in the case of congestion in the switch, returns a busy status so that the sender can retry), whereas class 3 provides no acknowledgement. Unlike class 1, there is no connection set up, and therefore, no connection overhead.

The FC-3 layer defines functions that involve multiple N-ports, such as striping (increasing throughput by the use of multiple parallel N-port to N-port links for one logical connection) and multicasting. We will not describe the FC-4 layer which defines mappings of certain standard upper layer channel and networking protocols onto the underlying layers.

### 6.2.3 HiPPI event building tests

High rate, parallel event building with the message switching approach has been demonstrated in the VME environment using the HiPPI technology [15]. The demonstrator, shown in Figure 11, consisted of a non-blocking 8 x 8 HiPPI switch, three VME-HiPPI interface modules [31] acting as sources with three more as destinations, and a RAID [32] VME master running a UNIX-like real-time operating system.

The VME HiPPI source and destination modules include a 25 MHz RISC processor and a HiPPI link interface based on commercial chip sets. Stand-alone firmware running on the RISC implements the primitive operations needed to send and receive data over the HiPPI links. The Source tasks running on the RAID use the source and destination modules as HiPPI I/O servers. The Generate task, running on the RAID, generates event fragment sizes for the sources, and the Control task assigns destinations and implements different event building data flow control scenarios; one “push” and two “pull”. Client-server communication was via the VME bus; because of limited VMEbus bandwidth and the use of only one RAID, no event data was generated in the RAID or copied across VME into the sources; therefore, the sources sent event fragments of the size generated in the RAID, but containing null data.

![Diagram](a) Software and hardware components of the RD13 VME HiPPI-based event builder test bench; (b) the “push” and “pull” event building data flow scenarios.
The minimum latency for a source to establish a connection and start data transfer was measured as ~50 \( \mu \)s and was dominated by the firmware overhead (the overhead for connection set up by the switch is less than 1 \( \mu \)s). Connection requests were queued if an output port was blocked by an already established connection. When all event fragments were of the same size and events were allocated to destinations in a round robin schedule, the aggregate throughput of the 3 x 3 event builder saturated at 120 MByte/s for large event fragments (memory speed and system bus bandwidth of the processor in the VME-HiPPI interface modules limited the sustainable HiPPI source transmit bandwidth to a maximum of 40 MByte/s). For small event fragments, the software and firmware overhead limits the load that can be applied to the switch.

Nevertheless, event handling rates of the order 10 kHz were achieved with event fragment sizes of the order 100 bytes (a typical Atlas level-2 extracted “feature” size). Taking into account that the RAID resources were shared by the client tasks of all the HiPPI sources and destinations, higher performance (20-30 kHz) should be achievable by migrating the client tasks into the RIO modules, or by using a faster processor.

When a random destination assignment algorithm was employed, or event fragment sizes were allowed to vary following different statistical distributions, connection requests had to queue in the switch for busy destinations to become free (blocking of switch output ports), and lower aggregate throughput resulted. More details can be found in [15].

The results confirm that this approach will be able to sustain the average rate of event fragment emission required in the ATLAS architecture. At the same time they demonstrate the important effect of overheads in limiting the maximum event rate that can be handled at the LHC, where average event fragment sizes are expected to be between 100 - 1000 bytes. The impact of overheads can be reduced in several ways:

- use faster processors;
- eliminate operating system and interrupt overheads by writing stand-alone software and avoiding the use of interrupts (see for example section 7.3);
- pack multiple event fragments into a “super-fragment” and build “super-events”;
- use a custom hardware engine for the source and destination data flow control [11];

### 6.3 Discussion on event building with message switching technologies

The demonstrator results confirm that the performance required for the ATLAS architecture can be achieved with existing HiPPI adaptors and firmware running on modestly powerful processors. In principle, Fibre Channel class 1 could be used to perform event building in a very similar way. However, in comparison with the HiPPI switch used in the demonstrator, currently available Fibre Channel switches suffer from relatively large class 1 connection set up and tear down overheads (~100 \( \mu \)s) which, as indicated in figure 10 makes them inefficient for high rate event building with small event fragments.

If Fibre Channel switches are to be used in the CMS architecture, where event fragment switching must operate at 100 kHz, class 1 connection set up overheads in future 1 Gbit/s switches need to be reduced to around 1 \( \mu \)s, or better, to achieve efficient use of the nominal bandwidth. In view of the complexity of the Fibre Channel protocols and the fact that class1 is not intended for sending short messages, it remains to be seen whether Fibre Channel switch manufacturers will implement such low connection overheads. As mentioned before, a possible way around the problem is to pack multiple event fragments to form larger messages, but this adds additional processing overhead and latency that may not be acceptable for level2 triggers.
An important question that should be answered before selecting a large message switching fabric (e.g. HiPPI or class 1 Fibre Channel) is whether connection set up handling is centralized or distributed. A centralized connection handling scheme would have to queue connection requests that arrive while it is busy handling a previous request. When the connection requests are randomized in time but the load is heavy, or when the connection requests from different ports are correlated (as for example they would be if the fabric was operated as a barrel shifter), the queuing may lead to a much increased average connection overhead, which in turn would degrade event builder throughput.

An upper limit to the event rate that can be built by an N x N switch with centralized (non-pipelined) connection request servicing is given by:

\[ R < \frac{1}{T_c \times N} \] (1)

where \( T_c \) = connection set up latency measured when the queue for connection set up requests is empty; e.g. for a switch using centralized connection handling with \( T_c = 1 \, \mu s \) and \( N = 256 \), we find a maximum event building rate of only 3.9 kHz.

Currently available Fibre Channel switches [33, 34] have been measured to have \( T_c \approx 50 - 100 \, \mu s \), and they probably use a centralized software-driven connection set up mechanism; they are therefore likely to be far from achieving the required 100kHz performance. However, no measurements have been made on existing Fibre Channel switches to determine the maximum aggregate connect request rate that they can handle. The requirement for partitioning of the data acquisition system has lead to the proposal to use multiple, small, independent switches in ATLAS [16].

However, because very large commercial switches are not available, a large switch (e.g. for the CMS architecture) would have to be built by cascading smaller ones. A large non-blocking switch can be constructed using a multi-stage Clos network of smaller non-blocking switches. Figure 12(a) shows a two stage delta network which is blocking (as shown by the competing connections in bold). In the Clos network we add a third stage of switching to provide alternative paths and so make the network non-blocking as shown in figure 12(b).

![Figure 12:](image)

(a) (b)

**Figure 12:** (a) a two-stage network suffers from blocking whenever connections compete for the same internal link; (b) the 3-stage Clos network is non-blocking.

The Clos network approach has been proposed [35] as a means of building large Fibre Channel event builders for the LHC using the existing commercial switches (which typically have up to 64 ports). With this approach the bottleneck for connection request handling is
partially solved also, because individual switches do not have to handle the aggregate connection request rate of the whole event builder. On the other hand the connection set up process becomes more complex because individual switches need to communicate with each other in order to find a free path through the 3-stage network. The switch manufacturer offers connection set up software which allows this to be done with a latency which is just three times the latency of one stage. The complexity of multi-stage connection management suggests that it will continue to be implemented (mainly) in software in the future, implying that a centralized (or not fully distributed) mechanism will be used in each component switch, thereby limiting the ability to handle high trigger rates.

In short, if a message switching approach is to be used for event building, it will be important to select a technology and a switch architecture employing a very low latency, distributed connection set up mechanism. The issues around the cascading of HiPPI or Fibre Channel switches in event builders need more study.

7 Parallel Event Building using Packet Switching

As explained in sections 2.5.2 and 2.5.4, a packet switching technology allows the use of semi-permanent virtual connections (SPVCs) to provide bandwidth on demand. Figure 13 shows how an event builder could be constructed in which SPVCs are defined at the start of a data taking run to fully interconnect all source and destination modules. Once created, these SPVCs would remain available until the end of the run. Since the SPVCs provide all required connectivity, the dynamic set up of connections and the associated overhead that was necessary for message switching is now eliminated. Fibre Channel class 2 or ATM cell switching are two possible technologies that could be used in this way. However, to the author’s knowledge, there is no native class 2 Fibre Channel switch presently available. We will illustrate the approach using ATM cell switching.

7.1 The B-ISDN ATM protocols

The asynchronous transfer mode using 53-byte fixed-length cells (as introduced in section 2.5.2) has been adopted by the International Telecommunications Union (ITU) as the underlying packet switching technology to integrate all telecommunications services and specialized networks into one common, world-wide infrastructure, known as the Broadband Integrated Digital Services Network (B-ISDN). The ITU’s B-ISDN ATM standards [4] define
an architecture that can scale with technology and is designed to be adaptable to efficiently meet
the requirements of different applications (constant bit-rate and variable bit-rate services, with
or without a real-time requirement). ATM has subsequently been actively developed by the
computer industry, under the coordination of the ATM Forum [5], for high performance local
area networking.

ATM is a connection-oriented, packet switched technology employing fixed-length cells
made up of a 5 byte header and a 48 byte data payload. The ATM protocol defines three layers:

The physical layer specifies several options for the physical media, bit-rates and coding
schemes. The preferred physical layer standards are SDH and SONET, as previously
mentioned, which use the bit-rate hierarchy of 155.52, 622.08, 2,488.32 Mbit/s. Physical media
include UTP-5 (category 5 unshielded twisted pairs for speeds up to 155 Mbit/s) and multimode
and monomode fibre. Other functions of the physical layer include ATM cell header delineation
and header error detection and correction.

The ATM layer defines the cell format and how cells are switched between links using the
8-bit virtual path (VPI) and 16-bit virtual channel (VCI) identifiers of the 5-byte cell header.
The ATM layer also handles the mapping of VPI and VCI into the new values to be used on the
outgoing link to identify the virtual connection to which the cell belongs. The 48 bytes of user
data payload are not protected by error detection codes at this layer, but the header is protected
by an 8-bit error detection and correction code (HEC). A 3-bit payload type identifier (PTI)
distinguishes between cells that carry user data and cells that carry various types of control and
management information used by the network itself. The cell loss priority bit (CLP) defines the
priority to be used in dropping cells in a congested network.

The ATM Adaptation Layer (AAL) adapts the ATM layer to the requirements of specific
classes of service. The ITU standards define 4 service classes (constant bit-rate and variable
bit-rate services, with or without a realtime requirement) for different types of application. A
specific adaptation layer protocol is defined for each service class. The AAL5 protocol is the
most appropriate for event building applications. The AAL5 packet can be up to 64kByte in
length, and is segmented into (and reassembled from) a stream of cells for the ATM layer. The
packet trailer contains a length count and CRC check for error detection, but error recovery must
be provided by the higher protocol layers.

The physical layer, ATM layer functions and most of the AAL5 layer functions
are
normally implemented in hardware. Figure 14 shows the maximum theoretical user data
throughput using the ATM protocol over a semi-permanent virtual connection multiplexed on
a 622 Mbit/s SDH link. The effective maximum user data throughput is 542 Mbit/s due to the
combined effects of SDH framing overhead and ATM cell header overhead. The saw-tooth
shaped line shows the variation of throughput due to the packetization in fixed length ATM
cells. For comparison the figure also shows the throughput that is theoretically achievable with
class 1 Fibre Channel for different values of the connection set up latency.

7.2 Using an ATM switching fabric for event building

Large ATM switches are constructed as multi-stage switching networks, typically built from
16 x 16 or 32 x 32 switching chips. A description of a typical multi-stage, multi-path,
self-routing switching fabric can be found in [36].

7.2.1 Congestion in ATM switching fabrics

Multi-stage ATM switching fabrics route multiple virtual connections over each internal link.
Cells travelling on different virtual connections therefore may contend for access to an internal
link. Contention is resolved by queuing cells within each switching chip until the internal link is available. If the queue becomes full the switch is said to be congested.

There are two courses of action that can be adopted when congestion occurs. The first is to discard cells, the second is to use a hardware flow control protocol on the internal link to hold off the arrival of further cells until sufficient buffer space becomes available. The strategy of dropping cells is usually adopted in large switches designed for telecommunications infrastructure, whereas some switches intended for LAN applications use the link-level flow control strategy.

In telecommunications switches the aggregate traffic of a large number of independent subscribers is assumed to be random (i.e. cells have random destinations and arrive at random times). The internal buffers of the switching elements are dimensioned so that the probability of congestion occurring under this randomized traffic is acceptably small (typical cell loss probabilities are $10^{-10}$ or less).

7.2.2 Traffic shaping

The natural traffic patterns generated by a “push” architecture event builder concentrate ATM cells belonging to a given event towards the single destination assigned to that event. If sources inject the event fragments at full link speed, the internal buffers of the switching elements quickly fill up and acute congestion occurs.

In an event builder based on the telecommunications type of switch, the congestion causes severe cell loss and the event builder is unusable. In the case of a link-level flow controlled switch, the flow control prevents cell loss, but the simulation results in figure 15(a) show that congestion causes the event building latency to grow non-linearly with the load on the switch [14]. When the switch is offered a load above a certain critical level, the throughput of the congested switch is insufficient to handle the offered load and queue lengths in the sources grow without limit. Better event builder performance could be achieved by minimizing congestion.

Internal congestion can be minimized (or even eliminated) by using the technique of traffic shaping. As shown in figure 15(b), for the flow controlled switch, traffic shaping

---

**Figure 14:** Maximum theoretical effective user data throughput as a function of message length for 622 Mbit/s ATM and 1062 Mbit/s Fibre Channel class 1.
improves the shape of the event builder’s latency versus load curve and allows the event builder to operate at loads of up to 70-80% typically. In the case of a telecommunications switch, the traffic shaping techniques bring the probability of losing cells down to the range for randomized traffic ($10^{-10}$) or even better, and in addition they give a similarly improved latency versus load profile. In fact, traffic shaping makes the performance of the event builder independent of the details of the internal switch architecture. Under these conditions the behaviour of the event builder (latency, queue lengths in sources and destinations etc.) is determined by the form of the statistical distribution of event fragment sizes.

Traffic shaping consists of modulating the traffic emitted by a source in two aspects; the first consists of controlling the rate at which cells are transmitted on individual virtual connections, and the second involves control of time correlations between traffic on different virtual connections. In an N x N event builder, rate division by a factor N must be applied to each virtual connection so that, when the individual cell streams merge towards the destination, their aggregate traffic will not exceed the available bandwidth at the output port.

Rate division is necessary, but not sufficient to avoid congestion. Due to the synchronization of the sources with the trigger, each of the sources may emit a cell to the destination at the same moment, causing “wavefronts” of cells to converge towards the destination. In a large (say 1000 x 1000) event builder, 1000 cells will arrive almost simultaneously at the last switching element before the output port. Since the switching element typically has an internal buffer of a few kByte, it is unable to absorb the wave of 1000 cells (53 kByte). Several proposals to smooth out the “wavefronts” by changing the time correlation between cell traffic at the sources have been simulated [36]. One technique adds a small random delay to the injection time of each cell, and another approach correlates emission of the cell streams from different sources to form a synchronous barrel shifter in which one cell is emitted per time slot.

Rate control on individual virtual connections is supported in all ATM interface chip sets, but features for randomizing cell injection times or organizing a set of interfaces into a cell-based barrel shifter are not generally supported. This approach therefore requires additional customized interface hardware, excluding the use of commercial interface boards. A third traffic shaping proposal (suitable for commercial interfaces) emulates a circuit switched barrel shifter [37] by using virtual connections without rate division and performing time slot switching in software (synchronized by a global interrupt).

![Figure 15: Event building latency as a function of trigger rate for a 1024 x 1024 event builder using link bit-rates of 640 Mbit/s with an average event size of 1 MByte; (a) for the case of a flow controlled switch without traffic shaping; (b) for the same switch with traffic shaping.](image-url)
Simulation of a “pull” strategy event builder, based a single flow controlled ATM switch used to handle both the level-2 and level-3 traffic in ATLAS has given good results [38]. Because of the local/global level-2 architecture the level-2 traffic can be handled, with acceptable latencies up to the full 100 kHz level-1 rate, without requiring the use of traffic shaping. For the level-3 traffic it was found to be sufficient to apply traffic shaping based on rate division only (which is simple to implement).

Another traffic shaping scheme chains the emission of event fragments from sources by circulating a “send event fragment” token among the sources [19]. Sources only transmit the corresponding event’s fragment when they have the token. Several tokens can be in circulation so that events can be built concurrently in different destinations.

7.3 An ATM-based event building demonstrator

Figure 16 shows a VME-based parallel event building demonstrator based on a 8-port, 155 Mbit/s ATM switch [39] of the telecommunications type (no link-level flow control). Custom-designed VME-ATM interface modules [40] are used as sources and destinations. They are based on a commercial RISC I/O (RIO) VME board [41], on which the AAL5, ATM and physical layers of the protocol are implemented with commercial chip sets [42, 43]. The event building protocol layers are implemented in software in the on-board RISC. The physical layer protocol used by the switch is the 155 Mbit/s SDH protocol over monomode fibre.

![Figure 16: The ATM-based event building demonstrator.](image)

A logic state analyser and a broadband tester [44] are used for debugging, analysis and performance measurements. In order to be able to load the switch with traffic on multiple inputs, a number of low cost ATM data generator [45] modules, controlled by a personal computer, have been constructed. The data generator is a memory in which any desired pattern of cells can be loaded and sent out via a physical layer interface. The desired cell pattern is calculated off-line and downloaded to the data generator.

7.3.1 Event building protocols

Figure 17 shows the event building protocol stacks in source and destination. The event building software is implemented in two layers; the upper event layer is technology independent and adapted to the underlying ATM technology by the event fragment layer. The software runs on the 25 MHz RISC processor and is layered directly onto the chipset implementations of the AAL5, ATM and SDH physical layers. For performance the software is implemented without operating system and uses semaphore polling in preference to interrupts (interrupts are only
used for error conditions). Because the software is distributed, its performance scales to very large event builder systems.

In the source (destination), the event fragment layer performs fragmentation (reassembly) of event fragments into (from) AAL5 packets. In order to minimize the required receive buffer space allocation in the destinations, the length of the AAL5 packets used can be chosen close to the average event fragment size (up to the 64 kByte limit supported by the ATM standard). In the destination, the event layer reassembles events from the received fragments. A number of algorithms for detection of event completion have been proposed [46]. The event completion algorithm used to make the measurements reported in the next section was based on the timeout principle.

7.3.2 Performance measurements

Figure 18(a) shows the measured maximum output of a source when pushing out event fragments with the software described above. The dotted line is the measured output including ATM cell headers and incompletely used cell payloads; it drops for small event fragment sizes because of various hardware and software overheads [47]. The dashed line shows user data throughput, and the solid line shows the maximum frequency of emitting event fragments. The source software could support high trigger rates (~80 kHz) with small event fragments (<2 cells), but software overheads limit the efficiency of utilization of the bandwidth (~25%) in this case.

Figure 17: The event building software is implemented in two layers above the hardware-supported ATM protocol stack.

![Figure 17](image1.png)

![Figure 18](image2.png)

Figure 18: Performance measurements; (a) for the source pushing fragments at maximum speed; (b) for the destination receiving fragments from 8 sources and building 2 events simultaneously using the timeout algorithm; (c) measured event building latency and rate for a 2 x 2 event builder.
The corresponding measurements for the reception of event fragments, shown in figure 18(b), exhibit less throughput at larger event fragment sizes due to a bottleneck in the current design of the receiver (which can easily be removed). The more complex receiver software makes a larger impact at small event fragment sizes, limiting the maximum fragment reception rate to ~30 kHz.

Figure 18(c) shows the measured event building latency and rate obtained in a 2x 2 event builder in which each source sends 1 cell alternately to each destination. For small event fragment sizes, the event building software in the destinations limits performance, while for larger fragments it is the data transfer on the links that limits the performance. If required, higher performance can be achieved by using faster processors, more destinations than sources, higher link bit-rates, or collecting data from several events into super-fragments which are then used to build super-events.

8 Conclusions

The size and required performance of the data acquisition and software triggering systems at the future LHC experiments exceed by far that of previous experiments. This paper has reviewed some of the on-going R&D into technologies and techniques that appear to be suitable for parallel event building at LHC. Architectural options, such as the choice of trigger strategies and algorithms, the use of phased event building and intelligent selective readout can reduce the required data movement bandwidth. We have seen that, in order to use the full potential of high speed link and switch technologies, careful attention has to be paid to details.

First, only a fraction of the link bandwidth is available to carry user data, the remainder being consumed by encoding schemes, and packetization overheads at each level of the protocol stack. Message switching technologies have a hardware connection set up overhead that, depending on its magnitude, may significantly impact the throughput achieved when sending short messages. The aggregate connection set up performance of such switches must be carefully evaluated for the event building application envisaged (e.g. centralized handling of connection requests would be inappropriate for implementing a synchronous barrel shifter, where all connections must be switched simultaneously). Packet switching technologies offer semi-permanent virtual connections, which avoid the connection set up problems.

In a push architecture, the concentration of event fragments from multiple sources into the destination leads to output blocking or internal congestion in the switch. Depending on the technology, the result is either that throughput is reduced and event building latency increased, or that data are discarded. These problems can be circumvented by an appropriate choice of destination assignment algorithm and by using traffic shaping. The traffic shaping technique is mandatory when using an ATM switch without flow control, but it is also beneficial with all technologies, allowing operation of large switches at high loads with still acceptable latencies.

Software overheads can easily dominate system performance and mask the advantages of high speed link and switch hardware, as we saw from the measurements made using the standard TCP/IP communications protocols in a UNIX operating system environment. The performance penalty of redundant protocol layers, unused functionality and operating system overheads can be avoided by developing optimized event building software layered directly onto the network adaptor hardware. Results from HiPPI and ATM (155 Mbit/s) event building demonstrator systems have shown that software-limited event building rates in the range 10-30 kHz can be achieved with 25 MHz processors running software partly, or completely, in stand-alone mode, and avoiding interrupts. A combination of faster processors and higher
throughput links (in the case of ATM) should permit still higher event building rates to be achieved.

Acknowledgements

I am most grateful to J.-P. Dufey for suggesting numerous improvements to the paper and assistance with the preparation of some of the figures. In addition, it is a pleasure to thank the many colleagues who provided me with information and/or enlightening discussions during the preparation of this paper; D. Abbott, E. Barsotti, E. van der Bij, F. Chantemargue, S. Cittolin, M. Costa, J.-P. Dufey, T. Ferrari, G. Heyes, I. Mandjavidze, L. Mapelli, B. Martin, R. McLaren, S. Munoz, M. Nomachi, B. Segal, R. Spiwoks, A. van Praag.

References


[21] F. Chantemargue et al. (RD11 collaboration), private communication; a similar set of measurements are reported in: C. Miron et al., “Fibre Channel Performances with IBM Equipment”, RD11-EAST note 95-05 (June 1995).


[34] The IBM 7319 model 100 Fibre Channel Switch, IBM Corp.


WWW for Physics

The field of "WWW for Physics" is covered by the 3 following papers corresponding to a series of 5 lectures presented at the CERN School of Computing:

1. World-Wide Web Technology by H.K. Lie (abstract only)
2. Publishing on the Web by B. Rousseau
3. Interfacing to the Web by M. Dönszelmann

World-Wide Web Technology

H.W. Lie
W3C, INRIA, France

Abstract

The origin of World-Wide Web lies at CERN, the European Laboratory for Particle Physics. CERN, with its large physics collaborations involving hundreds of scientist all over the world, needed a uniform and simple system to share and deliver information. WWW was invented.

The Word-Wide Web is rooted in the internet technology. The HyperText Transfer Protocol (HTTP) build on top of TCP/IP, allow clients to get documents from WWW servers. HTTP supports document typing, thus permitting format negociation between clients and servers. Documents are referred to by clients with Uniform Resource Locators (URLs) based upon a naming scheme that guaranties unicity. Hypertext documents are formatted with the HyperText Markup Language (HTML), an application of SGML, the Standard Generalised Markup Language. HTML, currently at level 2, is evolving. HTML 3, among other things introduces tables and equations. Hot topics are related to "structure versus presentation" issues where extensions to HTML and/or style sheets are under discussion.

The W3 Consortium was created to interest commercial investors in a common Web technology and standard. Its current technological aims are: security and payment schemes, protocols for replication and caching and future versions of HTML including style sheets and content labelling. More information, including the current World-Wide Web technology and its future developments, can be found on the Web server of the W3 Consortium under URL:

http://www.w3.org/

The server also keeps a copy of the reference code (library, server and linemode browser) available for everyone.
Publishing on the Web

B. Rousseau
CERN, Geneva, Switzerland

Abstract

Publishing on the Web is a complex activity for which the knowledge of the HTML language is only a very basic prerequisite. Web publishers have to compose good HTML documents that satisfy a number of constraints imposed by the WWW media. They must then interlink these documents to build webs with good readability and maintainability properties. During this process, publishers have to use and sometimes to program software tools, ranging from editors and converters to sophisticated documentation systems. A tour of the web publishing tasks is presented. At each stage, guidelines and automated tools are proposed, that may contribute to build webs of good quality.

1 Introduction

The World-Wide Web (or the Web) is progressively becoming the standard way to publish information in High Energy Physics. The fact that the Web has been invented at CERN is not surprising because it solves many problems met by large distributed physics collaborations to share and deliver information.

Publishing on the Web covers many applications:

- Provide general information, e.g. see [1]
- Present activities and experiments, e.g. see [2]
- Publish results and reports, e.g. see [3]
- Publish engineering documentation, e.g. see [4]
- Distribute internal project information, e.g. see [5]
- Gather information on a given subject, e.g. see [6]
- Advertise services, e.g. see [7]
- Provide software online help, e.g. see [8]
- Monitor equipment, e.g. see [9]

The purpose of this paper is to give an overview of the concepts, techniques and tools to build such webs.

2 The Web Publisher’s Workbench

Browsing the web is easy (despite the fact that it is also easy to get lost), but publishing is a difficult and highly technical activity that requires knowledge of:

- the HyperText Markup Language (HTML),
- how to compose good HTML documents, avoid pitfalls and apply the WWW etiquette,
- how to design readable and consistent webs, by connecting individual HTML documents,
- how to organise and maintain a web,
- various tools and programming languages.

Figure 1 summarises the objects and tools involved in the publishing process. Central to the publishing process are the HTML documents. Publishing on the Web means producing such
documents possibly using simple and/or complex tools.

- With *text editors*, publishers use HTML as a programming language, editing both markup and text. No syntax checking is of course performed.
- *HTML editors* begin to appear. They belong to two categories: *syntactic editors* have menus to help the markup process. Syntax checking and HTML preview are usually provided. WYSIWYG *graphical editors* allow you to edit and view at the same time.
- Automatic *converters* (sometimes called *filters*) can be used to generate HTML out of documents written in other formats. For instance, there are converters for most commercial word processing systems (see section 4.4).
- Common Gateway Interfaces (CGIs) are programs coupled with a WWW server and that generate HTML files dynamically upon request. CGIs may for instance be used to connect to a database, select data, process it and produce an HTML document (see [10] for detailed information about CGIs).

Once published, HTML documents have to be maintained, using the above mentioned tools, and possibly *maintenance tools*.

### 3 Composing HTML Documents

The precise HTML syntax will not be detailed here, the reader is referred to some of the numerous tutorials and reference manuals available on the web, accessible from [11], [12]. HTML is easy and simple to learn. But this apparent simplicity should not hide some essential
features that are often forgotten by Web publishers, and sometimes also by the designers of WWW browsers and tools.

3.1 The Essence of HTML

One of the main ingredients of HTML is device independence. The success of the Web is due to the fact that the documents you publish will be accessible world-wide, whatever the WWW browser, and whatever the platform where it runs (Unix, Mac, PC, ...).

In order to fulfill this requirement, HTML is designed as a structural, not as a presentation markup language. With HTML, the author marks up its text with tags to indicate its structure. For instance there are HTML tags to indicate that a paragraph corresponds to a heading, that another paragraph is a numbered item in a list, that a piece of text is emphasised, etc. Not only an HTML document is portable from one platform to another, but it is a real electronic document that may be analysed, indexed, converted and transformed by automated tools.

Presentation features such as fonts, layout, alignments, colors are not usually portable from one platform to another and you will not (or rather you should not) find HTML tags for this purpose. How an HTML document is displayed is always in the hands of the WWW browser. The way to represent a given HTML element is sometimes hard-wired within the browser, and is sometimes customisable by the user.

3.2 The HTML Syntax

The HTML language is governed by a very precise syntax [13]. An HTML document should strictly obey this syntax in order to be interpretable by all existing WWW browsers. If HTML is incorrect, the result is unpredictable. Browsers such as Lynx [14] and Arena [15] signal if an HTML document is not correct. HTML correctness is not enforced by the widely used browsers NCSA Mosaic [16] and Netscape [17], that are very permissive and accept illegal HTML. Examples of illegal HTML constructs are:

- Putting headings inside list items to get ‘big’ listitems
  `<UL><LI><H1> ...</H1><LI> ...</LI> </UL>`

- Lists inside Headings (to achieve the same thing)
  `<H1><UL><LI> ...<LI> ...</UL></H1>`

- Using paragraphs inside headings
  `<H2>...<P>...</P>...</H2>`

- Using UL and OL to achieve indentation
  `<UL>...<P>...</P>...</UL>`

- Character highlights that ‘span’ structural elements (to have to insert less tags)
  `<EM>...<P>...</P>...</EM> -> <EM>...</EM><P><EM>...</EM>`

To avoid potential problems related to incorrect syntax, any HTML document has to be validated. For this purpose, you may use an HTML editor such as HotMetal [18] or Symposia [19], that will not let you introduce illegal HTML. You may also test your documents with the Lynx or Arena browsers. They will signal incorrect HTML. The best solution is probably to use HTML checkers (e.g. Weblint [20]), or send your documents to a remote validation service (e.g. HALSoft[21]).
3.3 Device Independence

Each HTML browser shows HTML slightly differently, we have seen why in Section 3.1. It is thus tempting to write incorrect HTML because it looks good on Mosaic or Netscape. In addition, some browsers such as Netscape support HTML extensions that are not part of the HTML standard. These extensions are very attractive because they add presentation capabilities to HTML: background images, centering of text, choice of fonts, etc. Such extensions break the device independence principle because they are not interpreted by all other browsers.

The Web publisher should also consider plain text browsers such as Lynx. Even if it contains graphics, any HTML page should make sense and be readable on an ascii terminal. This is why the <IMG> HTML tag to inline images has an ALT (alternate text attribute) to specify the text to be displayed when graphics are not supported. If this alternate text is omitted, the document may not be usable when seen with a plain text browser. For instance, look at Figures 2, 3 and 4 to see what happens when text alternative is not specified.

3.4 HTML Semantics

Syntactically correct HTML may still be doubtful because if it does not follow the HTML semantics. A typical example is the use of the <H1> tag to get bigger fonts. The <H1> HTML tag is, according to the standard, devoted to defining headings. Software tools may use this tag to, for instance, automatically generate a table of contents. Another example is the use of the <PRE> tag, intended for preformatted text, to center text. Do not forget that if the text looks centered with your browser, your clients may have different settings (e.g. font size).

Figure 2: The Home Page of the ATLAS Experiment at CERN, as displayed with Netscape. An inlined image contains both graphics and a title.
3.5 Images

A wide majority of websites use inlined images heavily. Since a large percentage of the users do have image-capable browsers, this expectation is not unreasonable. However, it is important to consider a few items when designing a web page. We have already mentioned the necessity to provide alternate text for images (see section 3.3). Another problem is the transfer time of images: because of their size, and because each image implies one connection, images may take much more time to load than the text. This may be frustrating, especially for readers with slow
connections. To minimise this problem, the web publisher should:

- not use too many different images
- supply indirections for large images. One way to do this is to inline a reduced version of the image. A link can be added to this image, in such a way that selecting the image leads to the full size image. It is also recommended to indicate the size of the image, to help the user decide whether or not to load it.

Think about making your images transparent, because you don’t know what will be the background color of your client’s browsers (use e.g. giftrans [22]).

### 3.6 WWW Etiquette

There are a few conventions which will make for a more usable, less confusing, web [23]. As a web publisher you should make sure this applies to your HTML document.

- **Do sign your document.** This will allow readers to mail you questions and suggestions.
- **Do indicate its status and date of validity.** When was it last updated? Is it complete? What is its scope?
- **Do provide a meaningful TITLE.** HTML offers the tag `<TITLE>` to assign a title to each one of your documents. This title is not displayed in the main window of the browsers, but is considered as an identifier of your document. For instance, it represents your document in a hotlist (or bookmarks) or a history window. Automated “robots” that wander the web to build indices also use this title. Thus the title should be chosen carefully in order to reflect the contents of the document.

### 4 Building Webs

Creating valid HTML documents is an important issue, but the real web publishing work consists in connecting individual HTML documents with hypertext links to form webs. Numerous issues have to be considered. We will not detail here every aspect of building a web, but we will rather give guidelines for the main ones. In this framework, HTML documents are referred to as nodes, because we consider them as part of a web.

Some of the most important issues are listed below:

- design of the contents,
- human factors,
- structure of the web,
- structure of the nodes,
- size of the nodes,
- consistency among the nodes,
- ease of navigation,
- readability,

The first two items are not considered here. *Designing the contents* of a documents requires the determination of who are the readers and what do they expect, in order to provide a document that covers their need. The term *human factors* covers the design of a document best adapted for human perception.
4.1 The structure of Webs

The most central issue concerns the structure of the web, fully determined by the scheme used to establish hypertext links across the nodes. Figure 5 categorises several web structures with respect to their expressive power and their predictability. Expressive power measures the capacity for webs to express a lot of information with a minimal amount of text and nodes. Predictability measures the possibility for a reader to get lost when browsing.

The **sequence structure** is made of nodes connected with next/previous links. Its expressive power is very low but, of course, a reader will never be lost.

The **network structure** has the highest expressive power, because it allows any piece of information to be cross-referenced, thus avoiding redundancy. As a counterpart, networks may also be very unpredictable. Two problems have been studied in the literature. **Disorientation** happens when the reader cannot locate himself in the main course of reading because he is unable to build a mental model of the structure [24]. **Cognitive overhead** occurs in the process of reading hypertext which tends to present too many choices about which link to follow [25].

The **tree structure** is a good compromise between expressive power and predictability. Imposing a tree structure on hypertext is a method extensively used to solve disorientation problems. It is simple and easy to understand because it corresponds to the natural decomposition of a book into chapters, sections, subsections, etc.

For these reasons, we strongly recommend to use the tree structure as the main skeleton of your web. When more expressive power is needed, it is possible to superimpose a network structure onto the tree structure as shown in Figure 6 b. It may also be useful for the reader to be able to scan your web linearly, to quickly get an impression, or for a systematic search. This can be achieved by superimposing a sequence structure onto the tree (see Fig 6 a).

Disorientation may also be induced by webs incorporating many links to external webs. After some navigation, disorientation occurs because the reader may have crossed the “border” of the original web and cannot go back to the main discourse. A possible solution is to centralise
external links within a special index (see Figure 6 b). All links within your web are local, except in the index. For the reader, reaching an external web is in 2 steps: from a node to the index, and from the index to the external web.

![Diagram of web structure](image)

**Figure 6:** Superimposing a sequence (a) and a network structure (b) onto a tree.

### 4.2 Anatomy of Nodes

The skeleton of the web being defined as the superimposition of a tree, a network and a sequence, how should nodes be organised? A node is made of contents and of hypertext links that reflect the web’s structure. The structure suggested in section 4.1 implies that a node contains, in this order:

- A navigation panel (e.g. ascii or graphical navigation buttons) that allows:
  - hierarchical navigation to the root node of the tree, e.g. [Up] to go to the upper node and [Top] to go to the root node,
  - sequence navigation: [Previous] and [Next] buttons,
  - global access means, e.g. [Index], [Search] or [Contents],
- Indication of the context, to help the user to locate himself in the web. This can be for instance the title of the upper node and/or the title of the document.
- The title of the node.
- The contents of the node is made of text and images. It may contain hypertext links pointing to arbitrary other nodes of the web. These hypertext links are like cross-references; they support the superimposed network structure.
4.3 The maintenance problem

The life cycle of a web is similar to the software life cycle: the maintenance phase may be very expensive when it is not foreseen and organised from the beginning of the project. Maintenance may result in very expensive tasks when:

- There is redundancy. Changing the information in one place implies changing it in many other places.
- The web is moved, e.g. in another directory or another computer. This may result in manual editing or all the URLs in your web. In addition all links to your web will be broken.
- HTML files are renamed. Same problem as above.
- The layout of nodes is changed. Again this will end up with a lot of manual editing to change headers, footers, add images, etc.

To avoid these problems, design work has to be performed carefully. The maintenance, the evolution and the future versions of your web has to be foreseen from the very beginning. In this area, a primary recommendation is to look at other webs, to see how their publishers have organised themselves. Lists of “good” webs may be found on the WWW, for instance [26]. Another major recommendation is to automate the maintenance as much as possible: information is difficult to maintain as soon as it is in HTML form. Use converters, or home made scripts to generate it from non-HTML or semi-HTML formats, that are easier to maintain. Examples are numerous: scripts may be used to automatically insert standard headers and footers into your nodes, local names may be used instead of URL, the latter being inserted automatically from a URL dictionary (mechanism similar to BibTeX), table of contents may be build automatically from the heading tags, etc.

Among all the maintenance problems of the web publisher, the worst one is related to broken links. A link between a source and a target is broken when the target moves, or is renamed. Avoiding this problem is difficult, and there is no general solution. The web publisher may:

- use relative URLs within your web. When the web is moved without restructuring, these links remains valid.
- check the links with tools such as MOMSpider [27]. These tools will signal all broken links.
- use converters. With tools such as LaTeX2HTML or WebMaker (see section 4.4), the links are automatically generated each time a new conversion is performed. Thus all the links within the generated web are valid.

These guidelines will help you to maintain your links. They will not at all help people pointing to your web: external links pointing to your changed nodes may be broken. A number of tricks may be played with redirection nodes, symbolic links, CGI scripts, or server rule file, but they only apply when the web is moved without being changed.

A better solution relies on Local Resource Names (LRNs), explained in [28]. The idea is to define hypertext link targets with logical names. If unicity of these names can be assumed for the whole web, one of these name is sufficient to express a link to a given target. LRNs are used instead of normal URLs, they can be interpreted by a CGI script that is able to find the URL of a given logical name, by scanning a database of all URLs in the web. Since logical names are much less subject to change than the structure or the location of the web, it is very safe to point to web using LRNs.

A more ambitious attempt to solve this problem at the global level is the URN (Uniform
Resource Names) project. See [31] for detailed information.

### 4.4 Converters

A large fraction of the information published on the WWW is first produced using non-HTML formats. This explains why a large number of converters have been developed to automatically translate various formats to HTML (see [29] and [30] for exhaustive lists). Such converters exist for most word processors (see below), for programming languages (e.g. C, C++, Fortran, Lisp) and for many other formats such as man pages, excel tables, mail or newsgroups.

Of special interest for the web publisher are converters for word processors, that may be used to convert existing documents that have to exist in paper and WWW. Instead of editing HTML directly, it may also be a good idea to produce text with a word processor and to generate your web with a converter. The word processor files becomes thus the master out of which the web is generated. Table I gives a list of a few converters for some popular word processors.

<table>
<thead>
<tr>
<th>Word</th>
<th>• rtf2html [35]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• Internet Assistant [36]</td>
</tr>
<tr>
<td></td>
<td>• CyberLeaf [38]</td>
</tr>
<tr>
<td></td>
<td>• SGML Tag Wizard [39]</td>
</tr>
<tr>
<td>LaTeX</td>
<td>• LaTeX2HTML [33]</td>
</tr>
<tr>
<td></td>
<td>• HyperLatex</td>
</tr>
<tr>
<td>FrameMaker</td>
<td>• Frame2HTML [40]</td>
</tr>
<tr>
<td></td>
<td>• WebMaker [34]</td>
</tr>
<tr>
<td></td>
<td>• CyberLeaf</td>
</tr>
<tr>
<td></td>
<td>• FrameMaker5 [41]</td>
</tr>
<tr>
<td>Interleaf</td>
<td>• Cyberleaf</td>
</tr>
</tbody>
</table>

**Table I:** A few HTML converters for the most popular word processors.

Required features for a good converter are the following:

- conversion of text
- conversion of cross-references, footnotes
- generation of a navigation panel
- generation of TOCs and indices
- conversion of graphics, tables, equations
- multi-nodes web generation
- a configuration mechanism

Because of the major differences between how documents are organised on paper and on WWW, converting from the word processor world to the HTML world is not easy [32]. In general, paper documents are in one piece and read sequentially. By contrast, WWW documents are made of small interconnected chunks that can be read by following hypertext links. The capability for a converter to generate multiple interconnected nodes is essential. As seen in section 4.2, small nodes are easier to read and may be pointed to directly. Among all converters mentioned in Table I, only two satisfy this requirement: WebMaker and Latex2HTML. Other converters only produce one HTML file and are thus limited to the conversion of small source
Another essential property of a good converter is configurability. The problem of most word processors is that they focus on the presentation aspects of a document, ignoring semantics one. For instance in Word or FrameMaker, it is possible to create a style to specify that some paragraphs are numbered, are sized to 14 points and are followed by 5 cm space, but the fact that these paragraphs correspond to chapter headings (or to a list item, or to a normal paragraph) is not something known to the word processor. This is changing, especially with the advent of word processors based on SGML [45] engines. Nevertheless, a configuration mechanism is needed to instruct a converter about the semantics of each paragraph style, because this information is needed for HTML translation. WebMaker has a configuration language that not only can express the semantics of each paragraph style, but also to specify precisely how a given paragraph will be mapped to an HTML construct. This configuration language has shown to be an essential feature for the generation of good hypertexts, without constraining the author in the presentation and the structure of its FrameMaker document [32].

4.5 Towards Documentation Systems

HTML is a very interesting medium to deliver information. However, it is not the only one: documents often have to exist in multiple forms, depending of the way they are used. Paper is certainly unavoidable for a careful reading, while editable formats are needed when the document is evolving. When a document does not need to be distributed world-wide, other more powerful hypertext formats may be used, such as DynaText [44].

In addition, HTML is certainly not the best way to produce information because its markup is too poor to reflect all the semantics of text within a document. There is no way, for instance to express the concepts of a table of contents, of a glossary, of a piece of source code, of all those textual constructs that are typographically identified in a paper document. Given the current state of HTML and HTML editors, word processors are today the best way to produce documents.

Documentation system is a generic name to cover systems that clearly separate the production, management and delivery aspects of documentation, as shown in Figure 8. Production can be performed with standard word processors. The native documents can then be converted into a pivot format and stored into a database. This database supports a real management of the document in terms of requests, transformations, versioning, etc. Delivery is ensured by dedicated converters to generate postscript, pdf, HTML, or native editing formats.

Such futuristic systems do not exist at the moment. Nevertheless, one may observe an evolution towards this concept. The central element of documentation systems is the database. Several companies offer publishing solutions based upon databases:

- **NaviSoft** has developed NaviServer [42], a WWW server with an integrated DBMS. The database stores HTML documents and extract them dynamically when they are requested. Features such as automatic indexing, access control and version management are available. The DBMS may also be used to manage non HTML data, through a form interface.
- **O2 Technology** has extended its Object-Oriented DBMS with a WWW server [43]. Classes in the database can be associated with methods that give them the capability to represent themselves in HTML.
- **Electronic Book Publishing** [44] offers a modular publishing system that is very close to

---

1 LaTeX is an exception, mixing both structural and presentation information
Deciding the size of a node is a difficult issue and is a case by case decision. The solution is always a compromise between two rules:

- **A node should always address one well defined subject**
  because your readers may want to point to your page directly, and not through the root node of your web. If your node covers only one subject other webs may refer to it, thus becoming reusable.
- **Nodes should not be too long**
  Short nodes take less time to load, they favor direct access to information by avoiding scrolling and searching.

The presentation of the nodes should be consistent. Using the same style, logos, navigation devices throughout helps the reader to locate himself within the borders of your web, and avoid disorientation.
from the one described in Figure 8. Documents may be produced with various word processors, are converted to SGML and stored into a database. They may then be transformed into the DynaText proprietary hypertext format, or converted “on the fly” to HTML with the DynaWeb server.

![Figure 8: Architecture of a documentation system. Various word processing system may be used to produce documents. These source documents are converted in a pivot format (SGML may be a good candidate) and stored in a database, from which they can be retrieved and delivered in HTML, but also in other formats.](image)

5 Conclusion

In many respects, publishing on the web is similar to a software programming activity. The programming language is HTML. Its syntax must be understood and well used to be able to write portable HTML documents. Composing good HTML documents and building webs requires skills similar to those involved in software development, for instance: design, avoid redundancy, use a tree rather than a cyclic network, specify the author and the status, use a local naming scheme, automate tests and maintenance, stay short, are guidelines that make sense in both programming and WWW worlds. LRNs mentioned in section 4.3 take their inspiration from the well known software encapsulation principle: a web is given a logical interface that hides its internal naming scheme and implementation.

Today, the programming activity is helped with software development and CASE tools. Such tools are clearly missing in the web publishing world, where few tools exist beyond emacs and vi. Let us hope that tools will soon appear to help us designing, writing, assembling, testing, and maintaining webs of good and controlled quality.
6 Acknowledgements

Special thanks to Mario Ruggier, co designer of WebMaker, for his suggestions and his detailed reviews of the successive versions of this article.

7 References

   http://www1.cern.ch/CERN/GeneralInfo.html
   http://www1.cern.ch/Delphi/Welcome.html
   http://alephwww.cern.ch/ALPUB/pub/pub95-010-ppe_LEP_energy_cal/ecalpaper/ecalpaper.html
[4] ATLAS engineering information
   http://atlasinfo.cern.ch/Atlas/ENGINEER/drawing.html
[5] The MOOSE R&D project
   http://www.cern.ch/OORD/Home_oord.html
[6] STING - Software Technology Interest Group
   http://dxsting.cern.ch/sting/sting.html
   http://www1.cern.ch/PTTOOL/SoftwareTools.html
[9] SPS parameters online
   http://seattle.cern.ch/teletext/view110.html
    http://WWW.Stars.com/Vlib/
[12] HTML info in the Yahoo collection
    http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/
    http://www.w3.org/hypertext/WWW/MarkUp/html-spec/html-spec_toc.html
    http://www.cc.ukans.edu/lynx_help/Lynx_users_guide.html
    http://www.w3.org/hypertext/WWW/Arena/
[16] NCSA Mosaic Documentation
    http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/mosaic-docs.html
    http://home.netscape.com/eng/mozilla/1.1/handbook/
[18] SoftQuad HotMetal PRO
    http://symposia.inria.fr/symposia/Welcome.html
    http://www.khoral.com/staff/neilb/weblint.html
[21] HALSoft HTML Validation Service
[22] giftrans documentation
    http://www.cc.gatech.edu/cns/software/giftrans.1.html
    http://www.w3.org/hypertext/WWW/Provider/Style/Overview.html


[26] The Awesome Lists  
http://www.clark.net/pub/journalism/awesome.html

[27] MOMSpider Documentation  
http://www.ics.uci.edu/WebSoft/MOMspider/


[29] HTML Converters  
http://www.w3.org/hypertext/WWW/Tools/Filters.html

[30] HTML Converters info in the Yahoo collection  
http://www.yahoo.com/Computers_and_Internet/Internet/World_Wide_Web/HTML_Converters/

[31] WWW Names and Addresses, URIs, URLs, URNs, URCs  
http://www.w3.org/hypertext/WWW/Addressing/Addressing.html


[33] N. Drakos. All About LaTeX2HTML  
http://cbl.leeds.ac.uk/nikos/tex2html/doc/latex2html/latex2html.html

[34] WebMaker Documentation  
http://www.cern.ch/WebMaker/WebMaker.html

[35] rttfohtml - A Filter to Translate RTF to HTML  

[36] Internet Assistant for Microsoft Word  
http://www.microsoft.com/msoffice/freestuf/msword/download/ia/default.htm

[37] WordPerfect Internet Publisher For Windows  
http://wp.novell.com/elecpub/intpub.htm

[38] CyberLeaf Documentation  
http://www.ileaf.com/ip.html

[39] HTML Tag Wizard  
http://www.w3.org/hypertext/WWW/Tools/tagwizard.html

[40] Frame2HTML  

[41] FrameMaker5  
http://www.frame.com/PRODUCTS/Products.doc.html

[42] Navisoft Home Page  
http://www.navisoft.com/index.htm

[43] O2 Technology Home Page  
http://www.o2tech.fr/o2web/o2wwwgateway/o2www/info?TheService

[44] Electronic Book Technologies  
http://www.ebt.com/

[45] A Gentle Introduction to SGML  
http://words.hh.lib.umich.edu/bin/tei-tocs?div=DIV1&id=SG
Interfacing to the Web

M. Dönszelmann

CERN, Geneva, Switzerland

Abstract

Interfacing to the Web or programming interfaces for the Web is used to provide dynamic information for Web users. Using the Web as a transport system of information poses three constraints: namespace, statelessness and performance. To build interfaces on either server or client side of the Web one has to meet these constraints. Several examples, currently in use in High Energy Physics Experiments are described. They range from an interface to show where buildings are located to an interface showing active values of the On-line System of the DELPHI detector at CERN.

1 Introduction

Interfacing to the World-Wide Web[1] becomes necessary when we want to do more than distributing static documents. The following sections explain why we interface, where we interface and what the constraints are to interface to the Web. Two more detailed sections describe both interfacing on the server side and on the client side, showing the use of WWW in the field of High Energy Physics experiments[2]. A broader view of interfacing in general is presented stating which questions the Web interface designer should ask himself.

1.1 Why do we interface to the Web

Static documents are normally made available on the Web by an general http\(^1\) server. There is however information that you might like to put on the web for which you first have to build an interface (either on the server side or on the client side). Interfacing is necessary in the following cases:

- The information is available in a different information system than WWW (Databases).
- The information dynamically changes over time (Monitoring Systems).
- The information is readable via WWW but in an unsuitable format (Databases).
- The information has to be tightly coupled to an existing application (Help Systems).
- The browser is not capable of understanding the communication protocol or format of the information (Browser Extensions).

\(^1\) http stands for hypertext transfer protocol, the private protocol of World-Wide Web.
1.2 Where do we interface to the Web

Interfacing to the Web may be split up in two parts: server interfacing and client interfacing. You would typically use a server interface:

- to access other information systems,
- to access dynamically changing information,
- to keep clients simple and stable (there are less servers than clients).

A client interface would be used:

- to have an application interact tightly with a browser,
- to have a user interact heavily with a browser.

Either type of interface may be used:

- to convert information into the correct format,
- to have a browser upgrade its protocol and format recognition.

1.3 What are the constraints on interfacing to the Web

There are three major constraints when you design an interface to the Web:

- First of all you will have to find a namespace in the universe of information. This namespace is needed to make your information uniquely identifiable. We will see that the URL (Uniform Resource Locator) solves this problem in most cases.
- The second constraint is due to the Web protocol, called http (HyperText Transfer Protocol). Browsers use this protocol to request documents from servers. This protocol is stateless, which means that after information is transferred from server to client, neither of them remembers anything about each other. In other words, every request that arrives at a server has to be seen as a new request. The requests have no “state”, since the protocol is stateless. This poses restrictions on the interface design.
- Finally the performance might be a constraint. You could design an interface which works but which is incredibly slow. In this case people will use other ways than the Web to access the information. We will see that the choice between client and server interfacing mostly depends on the issue of performance.

2 Server Interfacing

A normal transfer of a file via the Web is shown in figure 1. The browser/client opens a connection to a server and issues an http request. The server looks up the file in its filesystem and sends it back to the browser. The connection is closed afterwards. The server in this case is no more than a gateway to the filesystem.
Let us see how to interface a server to a database. Figure 2 shows the dataflow in steps:

1) The user requests a form and fills it out (ordinary file retrieval).
2) The user submits the form, thereby activating a form-handling script.
3) The server queries the database.
4) The server replies to the user with a dynamically constructed HTML document.

The model used here is the same as for the ordinary file retrieval, except that the server now goes out to look in the database instead of the filesystem and that the server dynamically constructs a document.

Two common ways of interfacing information systems to servers are currently available. The first one uses the Common Gateway Interface specification[3], which makes it possible to
install portable scripts and programs onto a default http server. Figure 3 shows the dataflow for CGI:

- The user request some information (a filled out form is sent).
- The server calls a CGI script or program.
- The script or program accesses the underlying information system.
- The server adds a MIME[4] header\(^2\) and sends back the information.

![Figure 3: Dataflow for the retrieval of a document using CGI to couple the server and the script. The browser issues the http request to the server. The server starts a CGI script to access the information system. This script dynamically builds a page from that information and sends it to the browser. The server only adds a header to it.](image)

The three key constraints are met as follows: the namespace is mapped because CGI-scripts reside in one directory or are directly pointed to. This means that a request pointing at a CGI script uniquely identifies the information. The stateless nature of the gateway has to be taken care of by the script. The examples, that follow, will show how this is done. The performance of CGI-script coupling may be poor if the time to connect to the underlying information system is long. If that is the case a directly-coupled gateway may be the preferred solution.

The second way to interface is to directly couple your program with a standard http daemon. You basically replace two routines (HTRetrieve: which normally retrieves a file, and HTServerInit: which initialises contact with your information system). Directly coupled gateways have the advantage of a once-only initialisation. Figure 4 shows the details.

![Figure 4: Dataflow for a directly coupled gateway. The routines to access the information system are fully integrated into the server.](image)

\(^2\) MIME is used by WWW to describe the content-type and length of the document that is being transferred.
The rest of this section shows four examples of server interfaces to the Web. In each of the cases a typical problem is outlined and solved.

2.1 Example 1: CERN-MAP Gateway

The CERN MAP gateway makes it possible for a Web user to request the location of a building at the CERN site. The user provides the building number as input or points to a URL like:

\[
\text{http://machinename.cern.ch/building?34}
\]

Such a URL may be inserted into the CERN phone book interface. The gateway searches for the building number in a coordinate database, retrieves several maps and highlights the location of the building.

In figure 5 the result of the lookup of building 4 is shown. The smallest map shows the full CERN complex with the Meyrin site highlighted. The biggest map shows the Meyrin site with a rectangle highlighting what is shown in the medium sized map. The latter shows a ‘target’ on building 4. The three maps are glued together into a single bitmap.

\[\text{Figure 5: Result of the lookup of building 4 using the CERN-MAP gateway. Three separate views are presented: a detailed zoomed in view showing a target on the building number (top), a site (Meyrin) overview map showing the location of the detailed view (bottom left) and a CERN overview map showing the location of the site (bottom right).}\]
Looking in greater detail at the dataflow of this interface we see the following taking place, see figure 6: the user requests the location of a building, which the building gateway looks up in a coordinate database (this database has been created on beforehand from the PostScript files containing the maps). The building gateway returns an HTML document containing a reference to the map. This reference points to the map gateway and includes the coordinates of the building (not the building number).

The browser, upon receipt of this HTML file, will automatically request the unresolved map (in GIF format) by activating the map gateway. The request includes coordinates of the building, zoom factors, etc... The detailed map (Meyrin) is run through GhostScript, zoomed in on and a target is drawn at the specified location. The same detailed map is marked (unzoomed) with a rectangle. The overview map (CERN) is also marked with a rectangle. The three maps are glued together, converted into GIF format and sent back to the browser.

The three key constraints are met as follows: the naming scheme is satisfied since we use CGI scripts. The uniqueness is assured by the CERN building numbering scheme. The gateway has only one level of requests to resolve and is therefore stateless. Generating (bit-) maps on the fly is a time consuming task. Since buildings do not move very often we chose to cache the generated maps.

The gateway as described here may be useful in other applications where maps and drawings are concerned. Locations of computers, printers and other machinery could be shown.

---

3 The CERN building numbering scheme is based both on history and on functionality of the building. Some numbers are for physicist buildings and some for administration buildings (there are exceptions). For the CERN-MAP gateway it is adequate that no two buildings have the same number.
Faulty modules in a rack may be shown or electronics drawings could be highlighted with interesting points. In all cases the original drawings are the source for the information and markers and highlighting are added at retrieval-time.

2.2 Example 2: HIPE WWW Gateway

The next two examples are taken from the On-line Data Acquisition[5] and Slow Controls[6] System of the DELPHI experiment[7] at CERN. In the On-line System computers run monitoring processes to check the flow and control of the data acquisition system, the slow control values and settings, and the performance of the triggering system. Each of these processes set a state as the result of their inspections made. These states are monitored and controlled by state-management processes. The DELPHI on-line system relies heavily on network communications to interact between the state-managers, the monitoring processes and the DELPHI User Interface[8].

One way of communicating is via memory. HIPE (Human Interface for the Elementary Process)[9] uses shared memory to read values from slow controls (high voltage) channels. It is menu driven and displays channels, groups of channels and groups of groups in textual menus on a vt200 terminal. The HIPE WWW gateway[10] was designed to give people network access to the slow controls channels of DELPHI via the Web. It uses the core part of HIPE, but instead of producing textual menus it generates HTML. Figure 7 shows an example of the Outer Detector high voltage channel readings.

Figure 8 shows the dataflow for the HIPE WWW gateway. The Elementary Process reads the hardware and stores the information in shared memory. It is an autonomous process which is always running in the background. The HIPE WWW gateway, when started, associates with the shared memory and waits for a request from the browser. When this request comes in, a URL analyser looks what the user wants. Requests look like:

http://machinename.cern.ch/HIP/OD/function/hic_display/hv_channels/1/100

where OD equals Outer Detector, hic_display is the function to execute, hv_channels is the group of channels to show and 1 and 100 are the parameters to provide to hic_display. The URL analyser now calls the hic_display function which forces HIPE to read the channels from shared memory and compose a display.

The functions which used to provide menus and lists of channels on a vt200 display were replaced by function that compose HTML. This HTML is in the form of unnumbered lists instead of menus. The output, which is thus created, is a passive HTML file. HIPE WWW is aware which functions to execute when you select an item. The URL resolver basically re-composes URLs from these function pointers and inserts those URLs into the output. The end result is an HTML document showing values of slow controls channels, where each channel points to a function to explore that channel even further.
Figure 7: Web page showing the settings and readings of some HV channels of the Outer Detector of DELPHI. Each line shows the different attributes (voltage, current) of a channel. The lines are linked to pages showing more details of each of these channels. A reload of this page will show the same structure but all values would be updated.

The three key constraints are met as follows: HIPE WWW is a separate server and can thus use its full namespace. Each sub-detector is mapped to a separate area and functions are separated from menus. The two parameters in the URL (1,100) together with the URL resolver make sure that the interface is stateless. The HIPE WWW gateway can always construct a page of channels from a URL. No history is needed. Since the gateway hooks up to the shared memory once at start-up, performance is only limited by reading channels from memory. The whole setup turns out to be faster than the old HIPE vt200 interface.
2.3 Example3: DIM WWW Gateway

The DELPHI On-line System uses the DIM[11] system (Distributed Information Management) for its internal network communications. This system uses a publish-and-subscribe mechanism in a client server model. Monitoring processes publish information by name onto the network. Other applications, like user interfaces, subscribe to this information, and once subscribed are kept up-to-date by the monitoring jobs. The DIM system uses a central nameserver to tell clients where certain information (services) may be found. The system can be used for internal DELPHI running only, because both the central nameserver and the continuous connections do not scale.

To ensure that people at their home institutes have access to the DIM services a DIM WWW gateway[10] was created. The gateway works by doing on-the-fly conversions of so called DIM files. These files are basically HTML files, but contain extra DIM tags. When a DIM file is retrieved these tags are replaced by information (values) from the DIM system. Figure 9 shows the On-line System Status of DELPHI. Information like trigger rates, state of the data acquisition and slow controls is shown in a table marked up in HTML3.
Figure 9: A web page showing the Online System Status of DELPHI. The page shows a table in HTML3 in which DIM tags (not shown) were replaced at retrieval time with actual values of the Online System. Different parts, like trigger, data acquisition, LEP communication, safety and slow controls, are shown. Links from these parts to other DIM pages exist.

When the user retrieves a DIM file, see figure 10, the gateway searches for a converter from text/x-dim (the MIME[4] type associated with DIM files) to text/html (the MIME type associated with HTML files). The converter in its turn looks for <DIM SRC=""service""> tags. If found, it subscribes itself to the service and replaces the tag with the values it receives. It then releases the service again. All DIM tags are thus replaced by values and strings, resulting in a valid HTML document. This document may then be displayed by the user. If the user reloads the file, the same conversion takes place again, updating all values with more current ones.

Figure 10: Dataflow of the DIM WWW gateway. The converter is automatically called when a DIM file is accessed via the web. It parses the DIM file and replaces the DIM tags by information from the Online System. This information is retrieved by having the converter subscribe to the Online DIM services (and unsubscribe afterwards). A converted DIM file will be plain HTML, which can be sent to any browser.

The three constraints are satisfied in the following way: DIM files reside in the normal filesystem, so the gateway uses the standard namespace. The gateway provides only a one level interface, which certifies the statelessness. Conversions of DIM files tend to be slow, since the gateway needs to subscribe and release services all the time. By caching the actual service (the information is kept up-to-date by DIM) the performance is greatly improved.
2.4 Example4: EDWIN Gateway

High Energy Physics experiments are probably easiest explained by visualizing collisions. Specialized programs, like event displays or viewers, must be used in order to see these events from different angles and with different zoom factors. These programs are normally quite complex and difficult to use. To make event viewing available for the general public, especially for educational purposes, DELPHI undertook the effort to provide their event display via the Web.

EDWIN (Event Display WWW INterface) was designed to provide a generic way of interfacing event displays to the Web. The user uses an HTML form to request a certain event in a certain view (angle, zoom factor). The server provides a view in the form of a bitmap. The user may now iterate by re-submitting the form with different values, until he is satisfied with the result. This setup works but is very slow. Figure 11 shows an example of an event viewed via the Web.

![Figure 11: Web page showing two views of an event of DELPHI. The views are in gif format, included into a form. The values in the form may be changed by the user. Re-submitting the form will provide new views included in an updated form.](image-url)
When the user requests the first view of an event, see figure 12, the server allocates an event display to him and sends back an ID number in the next form. It stores this ID and the association to the event display in a “state” database. When the user requests the next view, the server looks in the database for the ID and thus knows to which event display the request has to go. Ordinary event displays are used, which redirect their (XWindows) output to a virtual X server. The EDWIN package, which is linked into the event display, grabs the window, converts it into GIF and sends it to the client. Internal communications between the http server and the event displays are done with the XTCP package[12], which provides a TCP/IP communication channel to an X client.

![Diagram](image)

**Figure 12:** Dataflow of the EDWIN gateway. A browser allocates an event display via a CGI script. The relation between browser and event display is kept in a state database. A request for a certain event is sent via the http server to the correct event display. This one in turn displays the event in memory, grabs a copy of it and returns a file in bitmap format to the server. The server wraps the bitmap into a new form, which is sent to the user and can be used to request the next event or a different view. The layout of this form is taken from a configuration file.

The key constraints are met in the following way, but with restrictions: Since CGI scripts are used, the namespace is taken care of. The interface is made stateless by sending back and forth ID numbers between the server and client. The gateway does not scale because of the number of event displays we may start. It is also not clear when an event display is free for use again. The performance turned out to be very cumbersome. Conversions and manipulation of graphics plus the transfer across the network make the interaction very slow. In rest of this paper we will describe client interfaces and re-evaluate how EDWIN could be implemented using client scripting.

### 3 Client Interfacing

Client interfacing has become necessary to satisfy some of the constraints the Web puts on interfacing. In the next sections we will discuss three types of extensions the user can make to his browser and how these extensions are supposed to be used. All three types of extensions satisfy the key constraints: the name-space is satisfied on the server-side and thus poses no further restrictions on the client interface. The statelessness is satisfied by a client side interface since the stateless character only has to be preserved between client and server. Between client and any extension a state-full protocol may be used. The performance is the key issue when deciding to put an interface on the client side or not. If a lot of interaction between the user and
the browser will take place it is worthwhile to transfer all data at once instead of transferring parts every time the user interacts.

3.1 Viewer Extensions

One of the easy solutions for a browser to handle unrecognized formats is to install separate viewers. The MIME header signals the browser what type of document follows. The browser retrieves the document and starts up a specific viewer for it, see figure 13. Viewers for formats ranging from PostScript and bitmaps to video are available.

![Figure 13: Dataflow for Viewer Extensions. When a browser retrieves a document in an unknown format it starts up a specific viewer for that format.]

You may invent your own MIME type, and ask the user to install a specific viewer for it. EDWIN could be implemented in such a way. The user would install a full event display and raw event data would be transferred.

The advantage of extending a browser with viewers is that it is relatively easy to add more formats and install more viewers. However, it does require the user to do this installation and the information provider may not be sure this is done. The viewer also loses connection with the browser after it is started, which means that there is no more interaction between the two. The viewer rests on its own and does not have any hyper-link capability. A new viewer is started for every document which cannot be viewed in the browser.

3.2 Connected Applications

A way to make interaction between viewer and browser possible is to use a connected application. In this case a connection is created and maintained between the extension (viewer) and the browser. A state-full protocol may be used here. Mosaic is the only browser that currently supports such a protocol called CCI (Common Client Interface)[13][14], which is not a standard yet.

CCI may be used in two ways. The browser can instruct an application (viewer) what to show, but the application may also tell the browser what to do, see figure 14. The latter was used by DELPHI to implement a context sensitive help system. Figure 15 shows a histogram presenter (much like PAW)[15] which starts Mosaic and retrieves specific help pages, when asked for.
Figure 14: Dataflow for Connected Applications using CCI. Three ways of communication between browser and application are shown. The browser may forward unrecognised formats to a viewer, an application may instruct a browser to retrieve a document or the application and browser keep each other informed on what each of them is doing.

Figure 15: The DELPHI histogram presenter overlaid with the Mosaic browser which shows a Help document. When the user clicks on an histogram, asking for help, the browser automatically retrieves the corresponding help document.
The communication may be truly interactive. One could think of an application where the browser trains a user about a certain program. While the user is going through the tutorial on the Web, the program follows and demonstrates how things will really look. If at a certain moment the user decides to try something in the program, the browser may follow what he does and point out what to do next.

Although CCI provides a two way continuous communication between browser and extension application, the publisher of the information still has to assume that the user has installed the extension application.

### 3.3 Browser Extensions

To avoid asking the user to install all the extension applications you could choose to use an extensible browser. This works in the following way, see figure 16. The default browser only knows a limited set of formats and protocols (HTML, text and http). If the browser now retrieves a PostScript file, it will also retrieve a PostScript extension module (a viewer for PostScript). It may then display the PostScript. The same can be done for unknown protocols. If a server should be accessed via ftp and the browser does not yet have the ftp module it may first retrieve it (via http for instance). The information provider should not only put his information on the Web but also provide the interpreter for his information.

![Figure 16: Dataflow for Browser Extensions](image)

The advantages are clear. The extensible browser is kept very simple. Any extension modules can be loaded when needed. The user does not have to configure any extra viewers, since the configuration is automatic. New formats and protocols may be introduced by the information supplier. The supplier of the browser can gradually innovate his product.

Instead of loading format and protocol modules one may also load modules that interact with the user. In the case of EDWIN the user may down-load Physics event data and the interpreter module for it. The user can then interact directly with that module and will be able to zoom, rotate and pan the event.

One of the first extensible browsers is available from Sun under the name HotJava[16]. It uses the Java language[17][18] to implement the extension modules. Java is like c++ but has some extra functionality to make it usable over the Web. Its main feature is that it is compiled into architecture neutral object code, which may be interpreted by the HotJava browser, running on any platform.
4  Summary

Interfacing to the Web needs a general approach. Figure 17 shows how data transfers across the Web and where one can interface. The programmer or publisher has to decide how much data he is going to transfer. He also has to decide how much the user will interact with the data and how state-full his data is. Taking into account who is going to use his information, what kind of browsers these people have and what kind of performance they expect, the programmer has to make a decision how and where to interface his information system to the Web.

![Diagram of Interfacing to the Web](image)

**Figure 17:** Interfacing to the Web. On the browser side extensions by module, viewers and connected applications using CCI are shown. On the server side, CGI and directly coupled gateways are shown.

The Web currently offers (albeit in a preliminary state) the following interfacing possibilities:

- Directly connected programs on the server side (*as shown in the HIPE WWW and DIM WWW gateways*).
- Loosely connected CGI scripts on the server side (*as shown in the CERN-MAP gateway*).
- Loosely connected Viewers on the client side.
- Connected Applications on the client side (*as shown for the DELPHI histogram presenter’s context sensitive help*).
- Integrated Extension modules on the client side, which are provided by the server (*an extension to be used in the future*).
References


Students

S.E. Agnvall, University of Stockholm, Stockholm, Sweden
A. Baldisseri, CEN Saclay, Gif-sur-Yvette Cedex, France
S. Bandyopadhyay, State University of New York at Buffalo, Amherst, USA
A.M. Baran, Institute of Nuclear Physics, Cracow, Poland
M.-L. Barja, CERN, Geneva, Switzerland
M. Beldishevski, INFN, Legnaro (Padova), Italy
D.J. Black, Fermi National Accelerator Lab., Batavia, USA
E. Blanco, University of Valladolid, Valladolid, Spain
M. Blume, Universität Wuppertal, Wuppertal, Germany
G. Bressani, University of Torino, Torino, Italy
C. Calomfirescu, National Centre for Health Statistics, Bucharest, Romania
A. Candiello, Dept. of Physics Galileo Galilei, Padova, Italy
M. Canepa, Università di Genova, Genova, Italy
M. Cattaneo, CERN, Geneva, Switzerland
J. Chudoba, Nuclear Centre of MFF, Prague, Czech Republic
S. Cohen, Institute de Physique Nucléaire, Orsay, France
T. Czarnecki, Physikalisches Institut, Tübingen, Germany
R.J. Dankers, National Institute for Nuclear Physics and High Energy Physics, Amsterdam, Netherlands
B. Debray, Grand Accélérateur National d’Ions Lourds, Caen Cedex, France
F. Delbecque, CERN, Geneva, Switzerland
O. Derugin, DESY, Hamburg, Germany
M. Dirkmann, Universität Dortmund, Dortmund, Germany
I.E. Falk, University of Lund, Lund, Sweden
H. Falkenberg, DESY, Hamburg, Germany
E. Fernandez Dominguez, ETSI Telecomunicacion, Madrid, Spain
I. Fernandez Gonzalez, Universidad Politecnica de Madrid, Madrid, Spain
S. Fernandez Vega, CERN, Geneva, Switzerland
M. Frank, CERN, Geneva, Switzerland
J. Fritze, CERN, Geneva, Switzerland
L. Gaido, INFN Sez. Torino, Turin, Italy
F. Giacomini, CERN, Geneva, Switzerland
M. Grippeling, CERN, Geneva, Switzerland
A. Gruber, Max-Planck Institut für Physik, Munich, Germany
P. Gunnarsson, Stockholm University, Stockholm, Sweden
P. Harris, Rutherford Appleton Lab. Didcot, UK
P. Huet, CERN, Geneva, Switzerland
P.M. Jonsson, University of Lund, Lund, Sweden
G. Kramer, ESRF, Grenoble Cedex, France
S. Kushpil, Lab of High Energy (NEEO), Dubna, Moscow, Russia
I. Laugier, CERN, Geneva, Switzerland
E. Leonardi, Università di Roma La Sapienza, Roma, Italy
C. Leroy, CERN, Geneva, Switzerland
M. Liebana, CERN, Geneva, Switzerland
O. Lodygensky, LAL, Orsay, France
A. Maier, Inst. für Hochenergiephysik der OAW, Vienna, Austria
A. Marcus, Tel-Aviv University, Ramat Aviv, Israel
F. Mauger, Lab. de Physique Corpusculaire de Caen, Caen, France
C. Meessen, CPPM, Marseille, France
S.B. Navert, Institut für Teilchenphysik, Zürich, Switzerland
D. Parena, INFN- Sezione di Torino, Torino, Italy
A. Patel, University of Sunderland, Sunderland, UK
I. Pertache, National Centre for Health Statistics, Bucharest, Romania
M. Ricard, CPPM, Marseille, France
S. Ritt, Paul Scherrer Institute, Villigen PSI, Switzerland
K. Rutz, Gesellschaft für Schwerionenforschung mbH, Darmstadt, Germany
A. Scortan, National Centre for Health Statistics, Bucharest, Romania
I. Semenova, JINR, Dubna, Moscow, Russia
M. Skiadelli, CERN, Geneva, Switzerland
P. Sollander, CERN, Geneva, Switzerland
I. Suchodolinska, Institute of Experimental Physics, Kosice, Slovak Republic
A.I. Syamtomov, N.N. Bogoliubov Inst. for Theoretical Physics, Kiev, Ukraine
G. Tel-Zur, Weizmann Institute of Science, Rehovot, Israel
J. Theissien, III. Physikalisches Institut Lehrstuhl B, Aachen, Germany
P. Uelkes, RWTH Aachen III, Aachen, Germany
J. Unger, Institut für Kernphysik I, Karlsruhe, Germany
F. Vassilopoulou, CERN, Geneva, Switzerland
H. Wagner, III. Physik. Institut A, Aachen, Germany
I. Younas, University of Southampton, Southampton, UK
Students

S.E. Agnvall, University of Stockholm, Stockholm, Sweden
A. Baldisseri, CERN Saclay, Gif-sur-Yvette Cedex, France
S. Bandyopadhyay, State University of New York at Buffalo, Amherst, USA
A.M. Baran, Institute of Nuclear Physics, Cracow, Poland
M.-L. Barja, CERN, Geneva, Switzerland
M. Beldishevski, INFN, Legnaro (Padova), Italy
D.J. Black, Fermi National Accelerator Lab., Batavia, USA
E. Blanco, University of Valladolid, Valladolid, Spain
M. Blume, Universität Wuppertal, Wuppertal, Germany
G. Bressani, University of Torino, Torino, Italy
C. Calomfirescu, National Centre for Health Statistics, Bucharest, Romania
A. Candiello, Dept. of Physics Galileo Galilei, Padova, Italy
M. Canepa, Università di Genova, Genova, Italy
M. Cattaneo, CERN, Geneva, Switzerland
J. Chudoba, Nuclear Centre of MFF, Prague, Czech Republic
S. Cohen, Institute de Physique Nucléaire, Orsay, France
T. Czarnecki, Physikalisches Institut, Tübingen, Germany
R.J. Dankers, National Institute for Nuclear Physics and High Energy Physics, Amsterdam, Netherlands
B. Debray, Grand Accélérateur National d’Ions Lourds, Caen Cedex, France
F. Delbecque, CERN, Geneva, Switzerland
O. Derugin, DESY, Hamburg, Germany
M. Dirkmann, Universität Dortmund, Dortmund, Germany
I.E. Falk, University of Lund, Lund, Sweden
H. Falkenberg, DESY, Hamburg, Germany
E. Fernandez Dominguez, ETSI Telecomunicacion, Madrid, Spain
I. Fernandez Gonzalez, Universidad Politecnica de Madrid, Madrid, Spain
S. Fernandez Vega, CERN, Geneva, Switzerland
M. Frank, CERN, Geneva, Switzerland
J. Fritze, CERN, Geneva, Switzerland
L. Gaido, INFN Sez. Torino, Turin, Italy
F. Giacomini, CERN, Geneva, Switzerland
M. Grippeling, CERN, Geneva, Switzerland
A. Gruber, Max-Planck Institut für Physik, Munich, Germany
P. Gunnarsson, Stockholm University, Stockholm, Sweden
P. Harris, Rutherford Appleton Lab. Didcot, UK
P. Huet, CERN, Geneva, Switzerland
P.M. Jonsson, University of Lund, Lund, Sweden
G. Kramer, ESRF, Grenoble Cedex, France
S. Kushpil, Lab of High Energy (NEEO), Dubna, Moscow, Russia
I. Laugier, CERN, Geneva, Switzerland
E. Leonardi, Università di Roma La Sapienza, Roma, Italy
C. Leroy, CERN, Geneva, Switzerland
M. Liebana, CERN, Geneva, Switzerland
O. Lodygensky, LAL, Orsay, France
A. Maier, Inst. für Hochenergiephysik der OAW, Vienna, Austria
A. Marcus, Tel-Aviv University, Ramat Aviv, Israel
F. Mauger, Lab. de Physique Corpusculaire de Caen, Caen, France
C. Meessen, CPPM, Marseille, France
S.B. Navert, Institut für Teilchenphysik, Zürich, Switzerland
D. Parena, INFN-Sezione di Torino, Torino, Italy
A. Patel, University of Sunderland, Sunderland, UK
I. Pertache, National Centre for Health Statistics, Bucharest, Romania
M. Ricard, CPPM, Marseille, France
S. Ritt, Paul Scherrer Institute, Villigen PSI, Switzerland
K. Rutz, Gesellschaft für Schwerionenforschung mbH, Darmstadt, Germany
A. Scortan, National Centre for Health Statistics, Bucharest, Romania
I. Semenova, JINR, Dubna, Moscow, Russia
M. Skiadelli, CERN, Geneva, Switzerland
P. Sollander, CERN, Geneva, Switzerland
I. Suchodolinska, Institute of Experimental Physics, Kosice, Slovak Republic
A.I. Syamtomov, N.N. Bogoliubov Inst. for Theoretical Physics, Kiev, Ukraine
G. Tel-Zur, Weizmann Institute of Science, Rehovot, Israel
J. Theissen, III. Physikalisches Institut Lehrstuhl B, Aachen, Germany
P. Uelkes, RWTH Aachen III, Aachen, Germany
J. Unger, Institut für Kernphysik I, Karlsruhe, Germany
F. Vassilopoulos, CERN, Geneva, Switzerland
H. Wagner, III. Physik. Institut A, Aachen, Germany
I. Younas, University of Southampton, Southampton, UK