428

Developing Complex Information Systems: The Use of a Geometric Data Structure to Aid the Specification of a Multi-Media Information Environment.

A. R. Warman, B.Sc.

A thesis submitted in completion of the requirements for the degree of Doctor of Philosophy

March 1990

The London School of Economics and Political Science

UMI Number: U047879

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

UMI U047879
Published by ProQuest LLC 2014. Copyright in the Dissertation held by the Author.
Microform Edition © ProQuest LLC.
All rights reserved. This work is protected against
unauthorized copying under Title 17, United States Code.

ProQuest LLC
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

BRITISH LIBRARY OF POLITICAL AND ECONOMIC SCIENCE

THESES

F

6692

x210260539

Abstract

Developing Complex Information Systems
The Use of a Geometric Data Structure to Aid the Specification of a Multimedia Information Environment.

The enormous computing power available today has resulted in the acceptance of information technology into a wide range of applications, the identification or creation of numerous problem areas, and the considerable tasks of finding problem solutions. Using computers for handling the current data manipulation tasks which characterise modern information processing requires considerably more sophisticated hardware and software technologies. Yet the development of more 'enhanced' packages frequently requires hundreds of man-years. Similarly, computer hardware design has become so complicated that only by using existing computers is it possible to develop newer machines.

The common characteristic of such data manipulation tasks is that much larger amounts of information in evermore complex arrangements are being handled at greater speeds. Instead of being 'concrete' or 'black and white', issues at the higher levels of information processing can appear blurred - there may be much less precision because situations, perspectives and circumstances can vary. Most current packages focus on specific task areas, but the modern information processing environment actually requires a broader range of functions that cooperate in integrating and relating information handling activities in a manner far beyond that normally offered.

It would seem that a fresh approach is required to examine all of the constituent problems. This report describes the research work carried out during such a consideration, and details the specification and development of a suggested method for enhancing information systems by specifying a multimedia information environment.

This thesis develops a statement of the perceived problems, using extensive references to the current state of information system technologies. Examples are given of how some current systems approach the multiple tasks of processing and sharing data and applications. The discussion then moves to consider further what the underlying objectives of information handling - and a suitable integration architecture - should perhaps be, and shows how some current systems do not really meet these aims, although they incorporate certain of the essential fundamentals that contribute towards more enhanced information handling.

The discussion provides the basis for the specification and construction of complete, integrated Information Environment applications. The environments are used to describe not only the jobs which the user wishes to carry out, but also the circumstances under which the job is being performed.

The architecture uses a new geometric data structure to facilitate manipulation of the working data, relationships, and the environment description. The manipulation is carried out spatially, and this allows the user to work using a geometric representation of the data components, thus supporting the abstract nature of some information handling tasks.

Acknowledgements.

I would like to thank all those friends, relatives and colleagues who have supported and encouraged me throughout the efforts described in this thesis. In particular I would like to thank my supervisor, Prof. I. O. Angell, for his tremendous support; not least in watching out for the split infinitives which I seemed to always include. Above all my thanks and love to Sue, who found time to become my wife.

Funding for this research was initially provided by the Science and Engineering Research Council. Later assistance was provided by the London School of Economics and Political Science.

Contents

Abstract: Developing Complex Information Systems i

Chapter 1: Information Systems and Enhancement Strategies 1

1.1: Introduction 1

1.2: The Problem Domain 3

1.3: Current Approaches 13

1.4: Suggested Requirements 21

1.5: Summary 29

1.6: Thesis Structure 29

Chapter 2: Current Systems 31

2.1: Overview of Current Systems 31

2.2: Management Systems 33

2.2.1: Project Management Systems 34

2.2.2: Resource Management Systems 38

2.2.3: Software Configuration Management 40

2.3: Development Systems 41

2.3.1: Software Engineering 41

2.3.2: SSADM 44

2.3.3: C.A.S.E. 45

2.3.4: I.P.S.E. 47

2.3.5: Analysis and Design Tools 47

2.3.6: Code Generators and Fourth Generation Languages 49

2.3.7: ISDOS 49

2.4: Data Handling Systems 50

2.4.1: Standard Text Description Systems 50

2.4.1.1: $\TeX$ 52

2.4.2: Graphic Presentation Systems 52

2.5: Idea Processors 54

2.6: Data Flow Control 56

2.7: Discussion of Current Systems 58

Chapter 3: Current Research 60
3.1: Overview of Current Research 60
3.2: Multimedia Systems 64
3.2.1: Example Systems 66
3.3: Hypertext 70
3.3.1: Example Systems 74
3.4: Hypermedia 79
3.5: Information Browsing, Presentation and Preparation 82
3.6: Information Structuring 86
3.7: Computer Architectures 90
3.8: Groupware 93
3.9: Summary 95

Chapter 4: Consideration of Current Systems and Related Research 98
4.1: Overview 98
4.2: Introduction 99
4.3: Discussion of Current Systems 100
4.4: Discussion of Current Research 105
4.5: Focused Systems and User Confusion 112
4.6: Conclusion 117

Chapter 5: Requirements for Genuine Information Systems: A Fresh Consideration 118
5.1: Introduction 118
5.2: Identifying System Components 124
5.2.1: The User and Presentation to the User 124
5.2.2: Control and Report 126
5.2.3: Integration 128
5.2.4: Communication 129
5.2.5: Processing and Computation 131
5.2.6: Data Groups 132
5.2.7: Data Basics 133
5.3: Suggested Requirements 134
5.4: Data and Information, Flow and Control 139

Chapter 6: The Information Environment 141
6.1: Introduction 141
6.2: The Information Environment 146
6.3: The Information Document 151
6.4: Designing an Information Document Architecture 152
6.5: Summary 154
6.6: Research Method 155

Chapter 7: The Information Environment and GENIE-M: Description, Generation and Control 157
7.1: Introduction 157
7.2: Geometrical Data Structures 160
7.2.1: Philosophy 160
7.2.2: Physical Description 161
7.3: GENIE-M Architecture 164
7.3.1: Modules and the Environment 164
7.3.2: Tile Transfer and Manipulation 165
7.3.3: Module Organisation within the Environment 165
7.4: The User Interface 166
7.5: Application Examples 168
7.5.1: Mixed Media Documents 169
7.5.2: Newspaper Layout Construction 169
7.5.3: Lecture and Presentation Control and Structure 172
7.5.4: Prototyping Methodologies 174
7.6: Problems with the Concept 174
7.7: Conclusion 177
7.7.1: Use of GENIE-M on Existing Systems 178
7.7.2: GENIE-M Users 179
7.7.3: Future Systems and the GENIE-M Architecture 180

Chapter 8: Design of the Prototype System 182
8.1: Introduction - Analysis and Design of the System 182
8.1.1: The Contribution and Incorporation of Software Engineering Techniques 183
8.1.2: Evolutionary Prototyping 185

8.2: The Structural Specification 187
8.2.1: The Kernel 188
8.2.2: The Server and User Interface 189
8.2.2.1: The Application Administrator 189
8.2.2.2: The Presentation Manager 190
8.2.2.3: The Event Handler 191
8.2.2.4: The Sub-Application Manager 191
8.2.2.5: The Geometric Tile Builder 192
8.2.2.6: The Tilestore Domains 192
8.2.2.6.1: The Localised Topological Tilestore 193
8.2.2.6.2: The Geometrical Tilestore 193
8.3: Hardware and Software Considerations 193
8.4: Summary 198

Chapter 9: Implementation of the Prototype System 201
9.1: Implementation Details 201
9.1.1: Implementation Concepts 202
9.2: Module Communication 203
9.2.1: GENIE_init(); 207
9.2.2: GENIE_send(); 208
9.2.3: GENIE_recv(); 208
9.2.4: GENIE_recvfrom(); 209
9.2.5: GENIE_stop(); 210
9.3: Module and Tile Architecture 211
9.3.1: ABORT 213
9.3.2: DEVICE 213
9.3.3: GROUP 213
9.3.3.1: GROUP ADD 214
9.3.3.2: GROUP DELETE 214
9.3.3.3: GROUP MOVE 215
9.3.3.4: GROUP RMOVE 215
9.3.4: SHOW 215
9.3.4.1: SHOW DEVICE 215
9.3.4.2: SHOW GROUP 216
9.3.4.3: SHOW TILE 216

9.3.4.4: SHOW TREE 216
9.3.5: TILE 217
9.3.5.1: TILE CREATE 217
9.3.5.2: TILE GET 217
9.3.5.2.1: TILE GET ATTRIBUTE 217
9.3.5.2.2: TILE GET DATA 218
9.3.5.2.3: TILE GET POSITION 220
9.3.5.2.4: TILE GET SIZE 220
9.3.5.3: TILE IDENTIFY 220
9.3.5.3.1: TILE IDENTIFY NEXT 221
9.3.5.3.2: TILE IDENTIFY ROOT 222
9.3.5.4: TILE MOVE 222
9.3.5.5: TILE READ 222
9.3.5.6: TILE RMOVE 223
9.3.5.7: TILE SELECT 223
9.3.5.8: TILE SET 224
9.3.5.8.1: TILE SET ATTRIBUTE 224
9.3.5.8.2: TILE SET DATA 224
9.3.5.8.3: TILE SET POSITION 225
9.3.5.8.4: TILE SET SIZE 226
9.3.6: TREE 226
9.3.6.1: TREE LOAD 226
9.3.6.2: TREE SAVE 227
9.4: Coding - the Sun Workstation Development Environment 227
9.5: Tile Space Storage Mechanisms 229
9.5.1: The 'matrix' module 229
9.6: GENIE-M and Distributed Systems 230
9.7: Summary 232

Chapter 10: Conclusions 234
10.1: Introduction 234
10.2: Review of the GENIE-M Philosophy 235
10.3: The Demonstration System 239
10.4: Successes of the Work 240
10.5: Limitations of the Current System: Threads for Future Development 243

10.6: Final Thoughts 245

Appendix 247

References 247

Error Codes returned by the Communications Routines 267

Record of Demonstration Session 269

Chapter 1.

Information Systems and Enhancement Strategies: An Introduction to the Problem.

Section 1.1: Introduction.

The origins of this work derive from the desire to seek a mechanism for implementing multiple editions of electronic books [Angell 86]. A specific example was considered suitable for general study, that of a computer-based text-book on crystallography. Multiple editions of the book would be generated on-line, where each different edition referred to a different crystal group. The majority of the text in the book would remain constant, but specific sections would require reconfiguration according to the space group and symmetries under discussion. In addition, a longer-term requirement was that multiple users should be able to access the electronic book(s), across a distributed multi-tasking environment. In order to provide the necessary capabilities, a totally new kind of electronic document structure was indicated. Analysis of the necessary design of this structure suggested a promising line of research, part of which culminated in this thesis.

Rather than generating a highly specific solution to the problem, it was considered sensible to consider the issues in more detail, such that a generalised definition might be derived. In particular, a structure technology based on geometry was suggested (for reasons discussed in section 7.2) as a possible foundation for the electronic books, but it soon became clear that such a concept had additional implications of a much wider-ranging nature. It was felt that the use of geometry for solving problems offers a number of benefits in terms of understanding and representing the computer-processing problems themselves.

Extending the geometric concept to more generalised cases represents a significant line of research. For example, the use of multiple geometric objects suggested the existence of relationships between them - at first according to the 'edition' of the electronic book, but subsequently in terms of the general application. Furthermore, since the geometrical objects could be viewed as related but independent objects, the way in which they could be used or accessed could also vary in correspondence with the application. A major first step was the observation that the use of a geometric architecture would promote easier definition of the problem and implementation of the solution. Standardisation on a geometrical foundation underlying a minimal architecture for computer-based information processing had the result that a multitude of enhancements were suggested for various applications.

This thesis is a report of the development work carried out for two related problems. Firstly, the development of a geometric structure to support the representation - and hence understanding - of problems and subsequent implementation of appropriate solutions, and secondly the generation of a computer-based architecture built upon these geometric structures, to be used for testing the contribution of geometry in implementing the solutions to the problems. At the same time, there is a parallel requirement for the development of a more formal methodology that can be used for applying the geometric structure to problem solving, since the provision of a tool alone would be of restricted value without an understanding of how the tool could be applied. The aim of the current work has not been to produce a 'Grand Theory' but instead to design and implement a minimal, primitive system to be used as the basis for prototyping further system extension. This in turn would be used as the basis for further work towards production of a larger, commercial system.

The work clearly required a comprehensive review of existing technology and techniques of problem representation. Simply patching geometry onto current application tools would not only be difficult, but also unlikely to result in clear major benefits. This would be due to the underlying concepts of the present application tools being founded on non-geometric principles, and so coming into conflict with the objectives of a geometric architecture. Therefore, the review was to be used in establishing from first principles the primary objectives of application tools, and their characteristic features. These would be used in developing the specification for the geometric structure, and subsequently the development of an architecture that implements those features.

Section 1.2: The Problem Domain.

The majority of modern computers have a fundamental feature in common: they are based on the Von Neumann architectural principle of single-state processing. Only one task can be in operation within the central processing unit of a single computer at any one time. There is almost no provision for the support of multiple computational tasks being shared among more than one machine in a distributed processing environment. Where such a facility is provided, it is typically by the use of support peripherals that provide interaction resources for the host machine to call on. The use of complex programs and fast hardware means that a computer may give the illusion of doing several jobs at the same time, but in reality this is achieved by swapping the machine's 'attention' from task to task in rapid succession.

A natural effect of this 'single-tasking' approach has been that both users of computer systems and developers of computer systems tend to work in a step-by-step fashion - considering each job (or each aspect of the current job) one-at-a-time. Such an approach has the advantage of simplicity, but the disadvantages of reducing performance and introducing complexity in the jobs to be performed. In particular, there seems little correspondence between this method of implementing computer-based information systems, and the practical realities of the problems encountered and requiring solution 'in the real world'.

As computers have become more powerful, the ease with which multiple tasks or applications can be accommodated within the machine has increased through the use of essentially low-level time-sharing models, and accordingly, the environment in which the user works has a greater potential to be more productive. An example is in the ability to run tasks 'in the background', for example printing operations; or in the provision of useful utilities packages such as on-screen clocks or mail arrival flags.

A comparatively recent innovation for workstations and personal computers is the graphic 'desktop', where users can see a pictorial representation of their jobs, all supposedly running simultaneously [Smith 82]. A simpler and more common variant of this is that each of the visible jobs is 'suspended', and only actually runs when the user is actively 'looking' at it.

This method of representing the current working state of the machine has proved very popular because of its initial simplicity for users, and similar facilities are apparently 'migrating down' from small workstations to be implemented on personal computers, as well migrating up to run on larger and more powerful computer systems. However, now that multiple activities can be seen to be taking place at any one time, many users wish to move away from rudimentary problem-solving by isolated 'hermit-like' processing of tasks, and pursue a more natural method of problem-solving, with the benefits of multiple tasks to communicate and share more information more rapidly. Such enhanced information handling requires complex system technology that does not as yet exist. The nearest current approximation is to use separate application tools in concert. This means that from the technological viewpoint, the user wants the practical ability to exchange data of varying kinds between the many tasks that can be seen running on a machine. But the less technological aspects of information handling involve going more into the higher-level or environmental requirements. In this case, users have needs going far beyond simple data manipulation; instead it becomes important to model as much as possible of the complete operational environment, in order to try and take into account real-world problem-solving circumstances which should influence the nature of information processing.

Few current systems incorporate environmental aspects into processing tasks, concentrating instead upon the technological safety of lower-level considerations such as the already mentioned data exchange between tools. In sales literature, a popular term for the passing of data in this way is 'Integration', but this often turns out to be an inaccurate exaggeration of the rather simple facilities offered. The low-level communications provided cannot offer high-level or complex facilities. Accordingly, weaknesses are generally found in the approaches used by many packages to implement integration. For example, while the ideal of integration would be to support data and environmental exchange, the actual implementation of integration usually offers less than this, resulting in only a rudimentary sharing or exchange of certain selected data. Indeed, such integration can in fact lead to additional complexity not only with regard to the technical issues, but also concerning the people using the systems, who must function while taking into account the practical, social and political consequences of their actions. Consequently, integration alone is insufficient to enhance computer systems to a point where they can better support problem solving.

More advanced objectives for enhancing computer-based information systems would imply that while the user's machine or machines may be running several distinct low-level tasks or programs, they should be able to share and exchange data in a meaningful fashion which reflect both the problem and its working environment, in effect requiring that the tasks being performed can be inter-connected and inter-related in some way, rather than existing and operating in isolation from each other.

Consequently, the tools should work with each other, cooperating in modelling the problem environment rather than competing in isolated and focused applications. All too often, the implementation of a computer-based information system gives little consideration to the actual nature or context of the problem; the continually changing environment of some application areas - such as most businesses - is typical in experiencing rapid changes. Avison points out in [Avison 85] that design of systems may be output driven, and yet:

> '... it is common and not unreasonable for users to require a change in the outputs even before the new system is operational. But because the system has been designed from the output backwards, such changes may necessitate a very large change in the whole system design. This can cause either a serious delay in the implementation schedule or an operational system that is unsatisfactory from the start.'

This could have a direct result in the form of increased difficulty in implementation or even failure of the project, simply because the application development process does not take into account the circumstances of the operational environment. The difficulties increase dramatically if several packages are used which were not originally intended to cooperate in problem solving.

Furthermore, the manipulation of problem data is a specific job that usually has to be initiated at the user's request, and cannot be performed as an automatic background task. Yet changes in the data may be the result of events occurring outside the user's influence, for example real-time events can take place while other work is being done, and any of these could be relevant to the tasks at hand. Currently, users cannot be clearly informed of an event, let alone invited to accommodate it into relevant work.

An example of this might be the preparation of a list of researchers attending a conference, for which the details are kept on a central database, but where the documents detailing events and names of those present are kept in a 'fluid' state up to and including the day of the conference. As the database is changed - possibly by several users - so the documenting system should immediately reorganise the underlying details appropriately without having to be specifically instructed to do so.

There are many ways in which current application development technology fails to meet requirements. A summary list of criticisms of typical computer applications is provided in [Avison 88]:

1) Failure to meet the needs of business. In particular, middle management and top management are being largely ignored by computer data processing. Management information ... is being largely neglected.

2) Another major problem is the inflexibility of the systems that are being implemented. ... Their designs are similar to those of the existing system. They are not intended to be adaptive: they are expected to parallel existing systems.

3) User dissatisfaction ... is a feature almost inherent in some computer applications. Frequently they are rejected as soon as they become operational. The user may agree the design of outputs with the systems analysts, but it is only when the system is operational that he sees the repercussions of his decisions.

4) There are problems with the documentation. The orientation of the documentation ... is towards the technologists and not the future users of the system.

5) Unusual conditions, commonly known as exception conditions, are frequently ignored in the computer system and this leads to incomplete systems.

6) A further problem is the application backlog found in data processing departments. Some users may have to wait two or three years before the development process can get under way and a further year before their system is operational. [There may be a] temptation ... to offer a 'quick and dirty' solution, particularly if the deadline ... proves to be somewhat optimistic.

7) Poorly designed systems will be difficult to maintain and in many businesses the maintenance workload is over 75% of total data processing workload.

Recognising some of these limitations in current systems, developers have tried to make their products easier to use, by sacrificing functionality in favour of operational simplicity. For example, in an attempt to provide some sort of assistance in the planning of which tasks need to be carried out, computer system developers have provided a facility to associate images with applications or very elementary concepts. This use of 'icons' has been promoted as the perfect solution to the difficulty of learning and subsequently coordinating the many tasks a user has to carry out, subject to the icon design meeting a number of criteria [Hakiel 87]:

1) Appropriateness - where an icon is rated according to how appropriate it is in relation to the desired function.

2) Comprehension - a measure of how likely it is that appropriate behaviour would follow interpretation of the icon.

3) Matching - where an icon is evaluated according to how well it represents a specific function.

For example, a word processing package that is not currently active might be represented on the screen as a small picture of a typewriter; and similarly, to dispose of some unwanted data (such as a file), a small picture representing the data can be moved across the screen to another icon representing a dustbin which will 'dispose' of the data.

In practice, however, the use of icons alone is inadequate, since they are simply different labels for the original objects - the only potential advantage they have is that users (and especially inexperienced users) can more quickly associate images with the corresponding concepts than they can link program names with the same concepts. Some people feel more 'comfortable' with icons rather than text.

This also assumes, however, that the user does not misinterpret the images used (which can often be very similar if they portray, for example, files of data), and in some cases the picture selected for the icon may have little obvious relevance or reference to the intended association. Indeed, there is rarely any consistency between packages with regard to exactly which icon image should be used to represent a specific concept - a problem aggravated by issues of copyright[1].

1: An example of this concerns the Apple Macintosh computers, where the Apple corporation are notorious for taking legal action against any firm imitating the Macintosh desktop or its Icons. This means that - for example - the 'trash can' symbol used to depict the means by which data/files can be removed from the system cannot be copied on other computer systems. The 'Next' computer from Next, Inc. uses a 'black hole' icon to represent precisely the same activities as the Macintosh trash can icon. Next, Inc. is a company set up by Steve Jobs, one of the founders of Apple.

Even at the comparatively low-level of command key sequences, we can find examples of inconsistencies between packages from the same source, such as the WordStar word-processors from MicroPro. On the WordStar Professional edition, the key sequence Control-Q and A is used to invoke the search and replace operation, whereas in WordStar 2000, the same sequence is used to exit from the current document, abandoning any changes made during the editing session, with possibly distressing results.

Until a genuine enhancement, or at the very least, a better integration of information processing systems is achieved, the current approach for developing computer-based information systems means that the potential for effective information handling, and support of the users, is minimal. Although raw data manipulation is well understood, more advanced issues which are relevant to problem solving, such as establishing the underlying and relevant data relationships, is more difficult to achieve than it needs to be. Furthermore, there is not necessarily any increase in inherent flexibility or power for problem solving even if multiple machines are interconnected. This is because, except in certain special cases, a computer system will manipulate data in a context-free fashion - with no real consideration as to what the data represents, or the unsuspected results that could be obtained from combining the various kinds and sets of data. Therefore, no assumptions can be made about what the data represents or how it is to be used. In many cases, the actual context of the data may even be specifically removed, despite the fact that this could be highly significant to the problem solving process. There is no generally applicable model which can be used for supporting distributed computation or support of problems.

A consequence of the current manner of information system implementation is that the user still has a major amount of work to do in deciding what operations must be performed in order to achieve the desired result - the computer will provide little assistance in representing or tackling the problem. Problem solving is therefore not greatly enhanced by the use of computer technology - at best the more trivial tasks can be performed more quickly. This is because significant improvement in terms of successful or efficient solutions can only be attained once the problem is well understood. Examples of the sort of difficulties which users can encounter are readily observed during the common tasks of preparing simple text documents. Although the computer can assist with mundane tasks such as storage of textual data (and possibly visual images if the package is sufficiently powerful); the author of the document still has the tasks of deciding the layout of material, preparing the correct formats for use with the output device, selection of the fonts to be used at certain points within the document, page size, line length, and many other jobs. It is true that some of the more advanced publishing packages used for the preparation of certain 'Fleet Street' papers offer some of these facilities [Honeywell 87], but generally, a single user still has to make the main presentation decisions. For all of these issues, the decisions must be influenced by the nature of the intended audience - thus requiring an understanding of the working environment. Some packages will advise the user that - for example - the attempted operation 'won't work' or 'won't fit'[2], because the tool simply is not designed to support it. However, at a higher-level, consideration should be given to the feasibility of operations that would be performed in the context of what may or may not be suitable under the given circumstances, thus taking into account the environment yet again.

Advanced users should be able to provide detailed requirements if they prefer, and so the computer would have different work to do according to the newly defined priorities and objectives; while novice users could use a standard or default description for the problem, for which the computer needed very little guidance but also offered reduced flexibility. We can describe a simple example which is representative of these issues by considering an enhanced form of a 'document', which must include data types other than simple text as standard components, in much the same way that sound and colour is a (vital) component of television or film 'documents' [Hall 87 & Harke 87]. Some computer-based systems - for example those based on hypertext concepts - do attempt to extend the document definition in this fashion, but remain limited for reasons which will be discussed in section 4.4.

In essence, the whole concept of document handling could suddenly become considerably more enhanced in nature, since the tedious work can - and indeed,

2: Although most users would probably prefer suggestions that will work.

should - be passed over to the machine, while the author or group can concentrate on the actual - and more important - content and context of the document itself which forms the value of the exercise. An extended document definition (as suggested by enhanced information handling architectures like that described in this thesis) would additionally make provision for automation of some significant parts of the work, such as incorporating the full effects of any content or context alterations by following through all the constructed cross-references and inter-dependencies.

A further detail concerns the use of devices. Computer systems are rarely 'standard' with regard to their peripheral or central equipment, and indeed any system that is fixed in such a way would be limited because of its inflexibility. Even those expansions which are supported are limited to a standardised connection mechanism, which limits the range of facilities that can be offered via a given peripheral device. In order to overcome the limitations, a diversity of peripherals are available. However, this can cause significant difficulties for users trying to 'install' packages for the first time because of the sheer complexity of choosing from the many options.

Assuming that a computer system can be instructed in the nature of the devices it will be using, there is no reason why information processing should not become far more context sensitive to both peripherals and environment. This would result in computer systems with a greater ability to assist in the practical effects of problem solving by modelling the circumstances according to realistic models of the users specifications.

At this stage, it would be easy to think that the problem has now been defined, in that we must establish a consistent architecture for manipulating data and relationships around and between various applications packages and systems, hardware or otherwise. However, while this would increase the potential for flexibility of information handling, on its own it does not immediately improve the underlying construction of the systems in general, nor does it enhance the range of capabilities available. Further thought allows us to perceive an enormous expansion facility within the framework of the stated ideas, offering the promise of significant developments leading towards more powerful computer-based information systems.

The hardware basis of computer and peripheral technology is advancing at a very fast rate, and the range of devices which could be inter-connected with the computer system is extensive and could form a very powerful configuration indeed. In this category of devices, we can list the various standard computer peripherals, such as monitor, printer (both impact and laser), keyboard and other input device such as a mouse and/or light pen; but we can also include devices such as scanners, voice and sound synthesis devices, telecommunications systems, video cameras, and video projection units (often used for echoing the display of monitor pictures). We can go further to include dynamically variable (analogue) devices which monitor or control real-time circumstances, such as lighting, temperature, sound and volume control mechanisms, or security devices such as motion detectors. However, by interconnecting such a range of devices we are introducing the potential for extreme complexity - some of which may be predicted, but in other cases may suffer from emergent problems. We are interested as much in studying the failures and limitations of such interconnection and enhancement, as in trying by experimentation to understand possible mechanisms for enhancing information processing tasks.

The connection of devices in a meaningful fashion, coupled with a computer's ability to carry out real-time updates or alterations to the data and the context of the data - as described in section 6.1 - would seem to be necessary in order to permit the construction of a considerably more advanced system. The design of an architecture suitable for such a proposed system has been listed in section 1.1 as one aspect of this research, and the work on producing an experimental system to test out these implementation ideas is described in this report. The overall objectives of this thesis involve several tasks. We must consider exactly what is meant by an idealised environment for information processing, why it is useful in problem solving - and indeed, why it seems to be necessary to enhance problem solving activities. This will require consideration of how various current systems offer selected components from such a concept, but do not yet achieve the full implementation of an environment. Finally, we describe in detail the work carried out in studying an architecture intended to allow users or group of distributed users to work within and become a part of the proposed Information Environment. Recognising that multiple users may require multiple configurations, we have to move away from imposing specific or focused answers and instead to experiment with developing an architecture that attempts to deal with the potential problems and conflicts that may be introduced.

Section 1.3: Current Approaches.

A key issue is to consider exactly what is meant by a computer-based system [Straub 89]. Typically, the result of labours by system developers will be a data manipulating system that is directed by the user within constraints or 'guidelines' determining what can or cannot be done with the package. The actual degree of user control over the system is a matter which has changed in parallel with the decreasing cost of improved computer and technology power. With the more capable equipment of recent times, users now - rightly - expect much more than a simple menu system to select from a constrained series of options [Wilson 85]. Current information handling systems are being sought which provide the more demanding applications that users are operating. Furthermore, the nature of applications is extending such that it is less of a viable proposition to develop isolated and insular packages. Instead, there is a real user demand for packages to conform with each other, to cooperate in the exchange and sharing of information.

The approaches being used to provide cooperation between systems currently fall into two categories. The first is to take a specific application area - for example, spreadsheet, database or word processing, and then to decide on how to make the system import or export the raw data to other popular or 'standard' packages, for example dBase III or WordStar. Typically, the only provision for this sort of activity is by the use of specific conversion utilities, that are barely integrated with the host package let alone each other.

The second and only marginally improved approach is to try and create a user environment that allows several unrelated packages to coexist successfully in the common environment, on the assumption that by 'consolidating' the operating circumstances, a package can be made to accept data from another package under the impression that it is in fact coming from another entirely different source. A simple example of this is found in the 'pipe' mechanism[3] used in the UNIX[4] operating system to 'build' connections between separate 'well-behaved' processes which have a consistent mechanism for data input or output [Bourne 83]; while more complex examples can be found in applications running under a windowing environment, and which support I/O redirection in order to enable multi-tasking and non-intrusive coexistence.

However, there is a distinctly separate area of development work which concentrates on determining at a very high-level each of the tasks that need to be carried out. In this category of information systems and development methodologies, we can include various Management Information Systems (MIS), Project Management Systems (PMS), Resource Management Systems (RMS) and development systems for software or general projects.

These information systems can be very detailed descriptions of guidelines to follow, or they may be presented as software running on computers that simply say what to do and when, but the essence of them all is to identify and isolate the goal and steps in reaching it, and then to provide control of the sequence of events that should be followed to achieve successfully a specific goal or goals.

The purpose of such packages varies, but ultimately they are intended to identify the areas where methodical use of technology or policy will improve or enhance existing procedures, whether it be the simple construction of a software package, or the development of a complete company organisation structure. The problem with this approach is that working with information is rarely an 'either or' situation. For any given problem with relevance to the real world, there could be a range of possible answers, depending on the environment of the issues. Social, moral, philosophical and political realities all dramatically reduce the number of problems which can have simple right or wrong, true or false answers. The ideal of the unified solution cannot exist - every technique produces a variety of possible solutions.

3: Denoted by the '|' symbol.

4: UNIX™ is a trademark of Bell Laboratories.

The division of packages means that it is obvious that there is almost no overlap between the fields of work. This is a major difficulty that is apparently a result of the areas being classified as 'territories'. The users who do the work have their application packages, while the managers who decide what work is to be done have their administration and directional packages. Research into the strategic aspects of information systems must inevitably take this into account. The territoriality seems to be in keeping with many of the popular organisational procedures, but fails to take full advantage of the information processing power and potential of computer-based information systems. Recognising the value of information as a commodity, and the benefits of enhanced, or at the very least more efficient, information handling, there must be a significant effort by organisations to move towards using the resources of the computer for the sharing and exchanging of data, although the degree to which this can be achieved depends in part upon human nature. [Clark 83] takes an idealistic view:

> 'Management by divisional competition and conflict must give way to cooperative management as information is shared rather than controlled.'

This straightforward and somewhat utopian observation has so far not been recognised or put into practice on any large scale, and placing aside the practical difficulties of incorporating such a philosophy into current information-consuming organisations, a major reason for rejection was clearly stated in the same article:

> 'The proliferation of computers increases the impetus for change, but system incompatibility hinders electronic reorganisation.'

The author then goes further in suggesting that:

> 'The usefulness of integrated software packages remains limited by general-purpose designs aimed at large markets.'

This latter statement seems to add a new aspect to the argument. The packages which are available may not in fact be general-purpose enough, because they are still based upon a rigid architecture and focused working environment, which specifically constrains the options which are available to users. Computer-based systems which are intended for information processing will at best support human-machine and/or machine-human interaction, whereas the ideal would seem to be human to human communication. Despite the great deal of effort expended on trying to make human-computer interaction as transparent as possible, there is still a definite overhead for using computers to communicate and to process information, and that overhead is determined by its effect on ease-of-use, performance, and usefulness. Only a few packages have followed appropriate and considered design strategies, as shown by the sometimes scattered and/or generalised options which are implemented. This means that the unspecific options may have subtle and unfortunate implications on each other. In very specific cases, extensive support tools may be designed and built which go some way towards promoting a suitable environment for face to face conferencing - such as the ICL POD. However, the problem with such systems is that they may be designed in a way that limits reconfiguration beyond the bounds of the original specification.

Rather than attempting to solve all problems with a single, but enormous, applications package, it might perhaps be preferable to concentrate on the difficulties of representing problems. If this is overcome, it may become possible to have any number of application systems to handle all possible jobs, knowing that they will each cooperate and contribute to the overall task objectives and their environment(s). A further advantage would be that this is in accordance with much of the emphasis of preferred management techniques such as those used in structured programming, where complex tasks are broken down into smaller and simpler problems. Each atomic problem is dealt with using a simple tool, and these tools or modules can be grouped together and invoked as and when necessary, knowing that the interaction between them is clear, well-defined, consistent and cooperative. However, the end result must be qualified by pointing out the delicate balance between risk and opportunity. More flexible systems are prone to complexity because of increased options, while constrained systems have reduced complexity but are limiting in their options. It is important not to lose sight of dangers and opportunities being associated with each other.

The work of this thesis suggests that the complex activities of everyday information handling cannot be successfully tackled using a single applications package, but instead require the use of a variety of task-orientated and dedicated tools which are fully and properly structured, thus providing for overall solutions. The tools are built upon each other and cooperate together in terms of a problem specification environment, that must have an enhanced capacity for providing problem descriptions.

Of crucial importance is the recognition that the human user is also a major environmental component that must be modelled. The user forms the highest level of information processing, positioned at the apex of the data and processing structures. If the users cannot work successfully with the system, then again the benefits or value of the system will be minimal. Therefore, there must be provision for, and inclusion of, Human and Computer Integration techniques, rather than the more widely promoted facility for Human/Computer Interaction. This in turn must be worked and tested in order to move towards the greater goal of using computer technology to support and enhance human-human interaction.

Ordinary interaction makes little attempt to incorporate user perspectives into the information processing tasks, assuming simply that the computer-based technology forms the peak of the presumed sole task of data manipulation, with the users as overseers and directors. There is even less support for the concept of the working group, contributing and functioning as a larger entity than the single user. A wide range of admittedly powerful, but low-level interaction tools (in comparison to the user's level of capability) are now available, but they all tend towards the erroneous assumption that the user must have complete responsibility for directing the integration and use of data within an application. This may often be valuable to a specific user, but it should be optional, rather than being imposed and enforced by the application tools and utilities.

Conversely, by modelling or 'integrating' the user as part of the information processing environment, the most powerful processing components (the user's mind and ideas) can be incorporated as part of the problem-solving system model to enable the provision of a much more cohesive, efficient and complete task execution capability.

But this must be extended further to support many users, supporting system enhancement on a larger and non-technological scale. Current work is often directed towards one user and one machine as a unit, possibly communicating in simple terms with each other as simplistic units. Such constraints inevitably limit the possibilities for work as a dynamic group, because the existing perspective is too focused, assuming black and white simplicity where facts and results are either true or false.

For computer-based information systems, the problem area is clear: application packages all too frequently take the form of directed, pseudo-generalised designs which cannot - by definition - match or be modified to fit precisely the individual needs of all problems and users. Compounding the error, system designers have focused their attention almost exclusively on the two fields of the application package and the administration package, all the while trying to make them general-purpose. Little allowance has been given to the issues of unpredictability within the environment - computer systems have a tendency to be 'special cases', which cannot be realistically handled in a generalised manner, nor necessarily dealt with by straightforward 'patching'. It seems that emphasis should have been placed on studying the problem environment, and the flow of information within and between the applications and the administration with a view to the end goal of enhancing problem-solving. In other words, it means recognising the fact that tasks lead from one to another at all levels, and further that these tasks can exist at differing conceptual levels, and so should not necessarily be performed by the same machine or even the same user. In effect, the focus on 'the task' should be widened to incorporate multiple users and indeed the working medium as being part of the information flow. Currently, the media support is viewed purely as a channel for input or output, rather than being integral to the function of the system.

This extension of the working environment model also introduces the issues of dynamism and feedback, for example concerning the reorganisation of systems both structurally and conceptually, modelling the environment as it also changes dynamically. An architecture for information processing must support dynamic reconfiguration, but permit users to over-ride and direct as appropriate. Current monolithic systems, which are focused and inherently inflexible, cannot achieve this.

Having recognised a preference for more dynamic systems, the realistic integration of packages by provision of an underlying architecture offers a stepping stone to enhancing computer-based information systems. The problem is therefore to work towards identifying ways in which computer technology can be used for enhancing the productivity of not just the single user, but also the working group, where the computer systems and distributed users (at all levels) are considered as vital contributing components of the team. Problems must be recognised, and solutions proposed and tentatively experimented with. Instead of concentrating on the technology itself as the only issue, it becomes important to study the processing components that seem to occur during information system processing, and thus begin to study how technology might be used to help in achieving enhanced work - if indeed I.T. needs to be used at all. For example, rather than merely mechanising the flow of data, technology should perhaps be used to improve the manner in which data can be worked - and this influences the nature of the architecture required.

Many sources have attempted to discuss this in detail, but some fail to take the important step of applying the ideas in a generalised form rather than in specific terms. For example [Stobie 87]:

> 'A key idea is that it (the integrated office concept) functions with a work group by helping the group members interact with each other more productively. This makes network mail a central feature of the integrated office.'

The above quote has faults in two areas. In the first case, the latter sentence of the quote cannot be implied logically from the first, as there is no valid basis for the conclusion.

In the second case, even if it could be implied logically, the latter sentence quoted is erroneous because it ignores the full implications of the first sentence. The provision of an electronic network mail (email) facility is very important to help with encouraging the flow of data between users, but is totally insufficient in itself to provide all the interaction that might be expected for integration, let alone provide an extensive problem representation facility. Email provision further assumes that the problem of confusion so often observed within message systems can in some way be eliminated or minimised.

For straightforward exchange of data or ideas, electronic mail has many advantages, including speed, accuracy, and reliability. But under certain circumstances these same factors can become unbalanced and result in passionate exchanges. Examples of this turbulence can be regularly observed when a 'public mail message' is sent to all mail users. Multiple recipients do not necessarily read their mail at the same time, and so replies and subsequent responses can quickly become out-of-sequence. They rapidly begin to suffer from a lack of clarity regarding the initial message that provoked the response. This is an inevitable consequence of the natural tendency for human communication to deviate from a specific discussion topic [Wilson 85b & Boyle 85].

Furthermore, the apparent 'insensitivity' of electronic mail can be a major problem. Comments which have a 'sarcastic' or 'humorous' intent can be interpreted in entirely the wrong way when read without the benefit of the accompanying body-language to indicate that the statement should not be taken too seriously. In conjunction with delivery delays, this leads to the acrimonious and highly aggressive series of message exchanges known as 'flaming' [Kiesler 89]. Such is the difficulty of email misinterpretation, that a series of symbols have evolved to help portray the 'emotion' of the message[5]:

Happy/Joke	8-)
Sad	8-(

All this leads us to conclude that instead of simply providing more capability for raw communication, consideration must be given to the circumstances under which communication is to take place, as it is this which will help deal with the difficulty of misinterpretation. Ultimately, it is necessary to examine the objectives of

5: To understand the symbols, turn this page clockwise through 90 degrees. The character symbols resolve themselves into frowning and smiling 'faces'.

communicating data around and between both the users and the machines working in the context of the task environment. Specifically, it is important to consider the mechanics of the communications in the light of identifying what such data movement is expected to achieve, and this in turn implies a requirement for an architecture that allows you to detect and hopefully to correct an imbalance. Only by questioning the purpose of an information handling tool - such as email - will we be able to implement a more useful facility that is of genuine benefit to the users.

Section 1.4: Suggested Requirements.

There are a number of properties which a more advanced strategy for information processing ought to have, in order to operate in a more flexible fashion. The first of these has been briefly mentioned in section 1.3, and concerns the internal construction and communication methods of the data manipulation. In other words, an architecture must be developed that supports the manipulation and interaction requirements by virtue of having been designed for that purpose, rather than taking existing structures which are then 'patched' by forcing changes in order to try and meet the refined needs. Once such an architectural specification is achieved, a more flexible system can be built upon it that supports much greater degrees of enhancement between the applications processes.

The second property involves presentation of the architecture, which requires an understanding of the natural human preference for working with images or pictures as descriptions of concepts which actually mean something, rather than with the textual or static graphic tokens (icons) that attempt (poorly) to portray those same concepts by simple relabelling. Using an enhanced internal architecture, it should be possible to support user-controlled dynamic portrayal of images. It is important to recognise the advantages of customisable dynamic depictions, because they support the idea of incorporating aspects of relationships into data control and presentation tasks. This follows because the data that is being worked with can remain consistent, but the presentation circumstances (which are under the user's direction) can vary in real-time.

In other words, not only can the data itself be presented in a useful manner under user control or preference, but also - and just as importantly - the relationships that exist between the data can also be incorporated, which may be important because they can change the way in which information is perceived and therefore evaluated. This seems to be a vital but frequently overlooked component of the way in which data presentation by computer can influence human thinking.

The depiction of the relationships could be enabled in many ways, but this thesis considers a first step towards the problem by experimenting with a prototype implementation of an architecture that uses geometrical and topological presentation systems, the exact mechanism of which will be dealt with in section 8.2. The adoption of such a geometrically-based paradigm would ideally encourage the users to explore the potential for benefits to be derived from studying the relationships that can be created between data objects, as well as providing facilities for an extended document architecture.

Such an approach is significantly different from the current emphasis on purely algebraic interaction, because it is not limited to the ordinary form of relationship depiction by explicit rules. Instead, relationships can be further supplemented by the inclusion of representation through geometric attributes. In order to explain this, we can consider all data as being stored within an unlimited cache area called a data space. We can then give a new attribute to data, which is its geometrical relationship to other data existing within the same data space. Having done this, we can now position associated data within the data space such that some of the relationships between data can be inferred from the grouping of the data (see Figure 1). In the diagram, the five blocks of data have no connection depicted in any form, yet instinctively most readers would group the blocks into two sets, A and B, by virtue of the proximity of certain of the blocks.

In systems that could enhance representation by use of geometric attributes such as that just described, a processing component would be responsible for performing the evaluation of relationships. The process could focus on computational components within the system, but by extending the concept of the working environment, we can

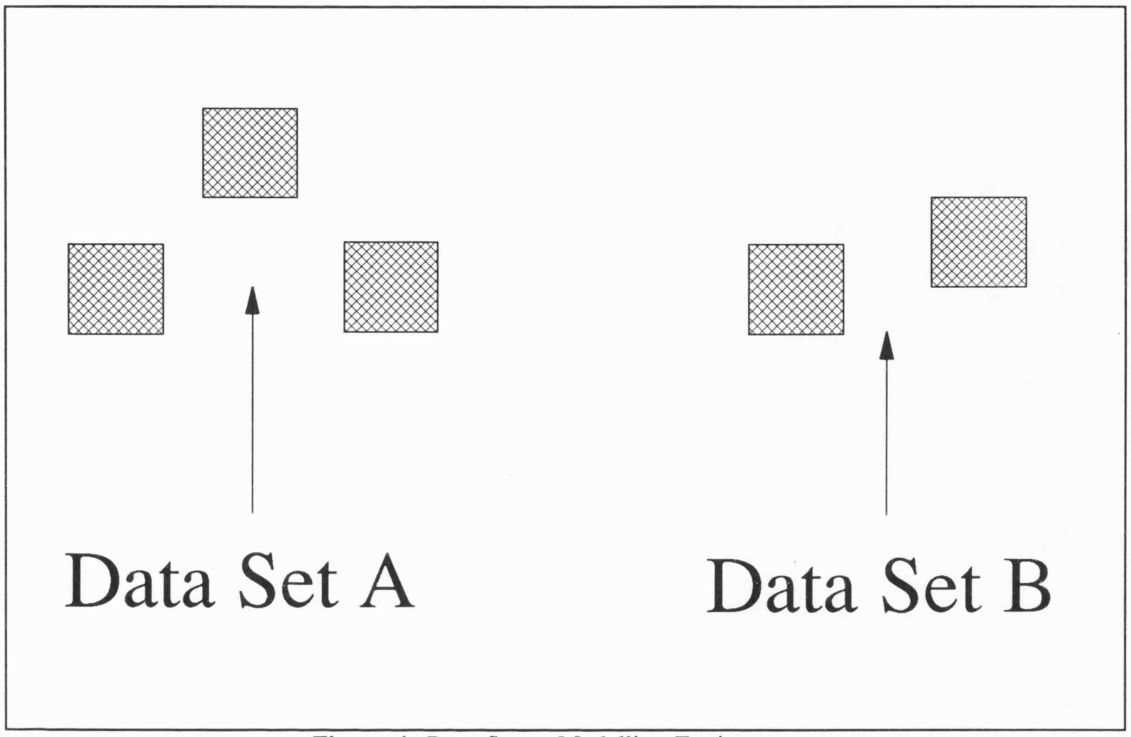


Figure 1: Data Space Modelling Environment.

include other components, such as the users themselves. The users are essential to the information processing tasks, and so should be recognised as such and be incorporated into the model. The relationships represented within the data space would be identified in a context-sensitive manner by the evaluating process. In other words, the geometric property alone can be interpreted to establish some of the nature of the data relationships. The interpretation itself may vary, depending on the tools being used to interpret the relationship itself. This is a valid proposition, because we can observe everyday examples of it in human users. We understand a data presentation only once we have learned to recognise how strategies can be used to depict a relationship, and hence identify which strategy is being used in a specific instance. Consequently, if a different strategy is used for depiction, then this can easily result in different interpretations.

Extending the idea of relationship depiction, it is not only the physical relationship between data that is of importance in establishing the association, but also the facility for supporting geometrical or topological relationships. In other words, we are not emphasising the absolute positioning of data as being the sole requirement for relationship specification, but also the attributes of relative positioning of data as contributing to the relationship specification through geometry. This enables us to work within the unlimited (virtual) volume of the data space, thus distributing the data in both a conceptual sense - within the data space - and also in a physical sense by relocation of data storage amongst multiple distributed machines. Taking the abstraction a stage further, we can remove the geometric component to address the relationships in purely topological terms, thus absolving any physical (machine) or geometric dependencies. In effect, we can propose the use of topological mechanisms to represent relationships explicitly, while geometrical mechanisms could still be used for the depiction of implicit relationships, which cannot otherwise be described.

Building a data handling system that can associate value with the relationships between data, using a topological viewpoint, introduces a number of significant advantages for the construction and operation of information systems. For example, altering the presentation of data implies only that the presentation technique itself must be altered, and not necessarily that there must be any modification to the data. No additional data processing is required, it is simply 'seen' differently.

Changing the presentation technique within a topological relationship could be accommodated by altering the association that exists between one set of presentation techniques or characteristics, and another set. This could be achieved in turn by redefining the geometric relationships that are depicted within the model. In effect, the relation between a data set and presentation system is removed, and a new and different relation with an alternative presentation system is then established.

A preferred advantage of this would be that the basic data being presented should not need to undergo drastic conversion or manipulation in order for the alternative presentation to be performed. Consequently, individual users could be catered for without requiring such extensive manipulation by the installed system. Instead, it would only be necessary to use the 'new presentation technique', which would be installed by establishing a new relationship between the original data and the new presentation system, as well as permitting the establishment of novel inter-relationships.

Using the more normal algebraic specification of relationships would require that a complete new algebraic specification is defined in order for the changes to be implemented. Such a procedure has the inherent risk of introducing paradoxes or inconsistencies within the algebraic rules.

A further advantage of the geometrical approach is that connections can be made between the working concepts or data sets at a high level of visualisation, without being concerned or restrained by the actual implementation of such connections. This follows because the relationship depiction matches closely some suggested models of information handling, where ideas are developed using relationships as well as the data itself, without necessarily requiring any understanding of how the relationship depiction is actually carried out.

Inevitably, there are limitations to such an approach. It is almost certain that there are circumstances which cannot be predicted or modelled in this fashion, and it is partly to experiment with these that the prototype system is being developed. The nature of the proposed architecture is inherently complex owing to the potentially large number of tasks which would be actively processing relationships at any one time, and also due to the dynamic nature of the multiple paths of communication between those tasks. Consequently, the architecture would minimally require a host machine which offers more advanced computer technology, such as that provided by multi-user workstations.

The last important property for a more advanced information system would seem to be the requirement for an ability to build up the representation of the working circumstances as a description of the Information Environment. The nature of this description could be considered as a form of 'document', with each such document detailing an electronic abstraction of the specific Information Environment. This point will be considered in section 6.3, but for the moment it is sufficient to note that the successful construction of such a description depends on the first two properties being implemented correctly.

At this stage, it may not be apparent why there is the necessity to include human users as part of the Information Environment. The justification for this context-sensitivity can be seen by considering a typical application of a computer - for example, processing information using a spreadsheet. At some stage, the computer user will wish to produce a hard-copy output of the results obtained from the spreadsheet. Currently, most application packages have to be 'installed' to recognise precisely which devices - in this instance, the printer - are attached and available for use. Without this data, the output would be severely restricted in its content or flexibility, assuming that it is possible to output anything at all. Thus, the efficiency of the computer depends in part upon the accuracy of its specified configuration at the time of use, and this configuration must be built up with the help of the human user.

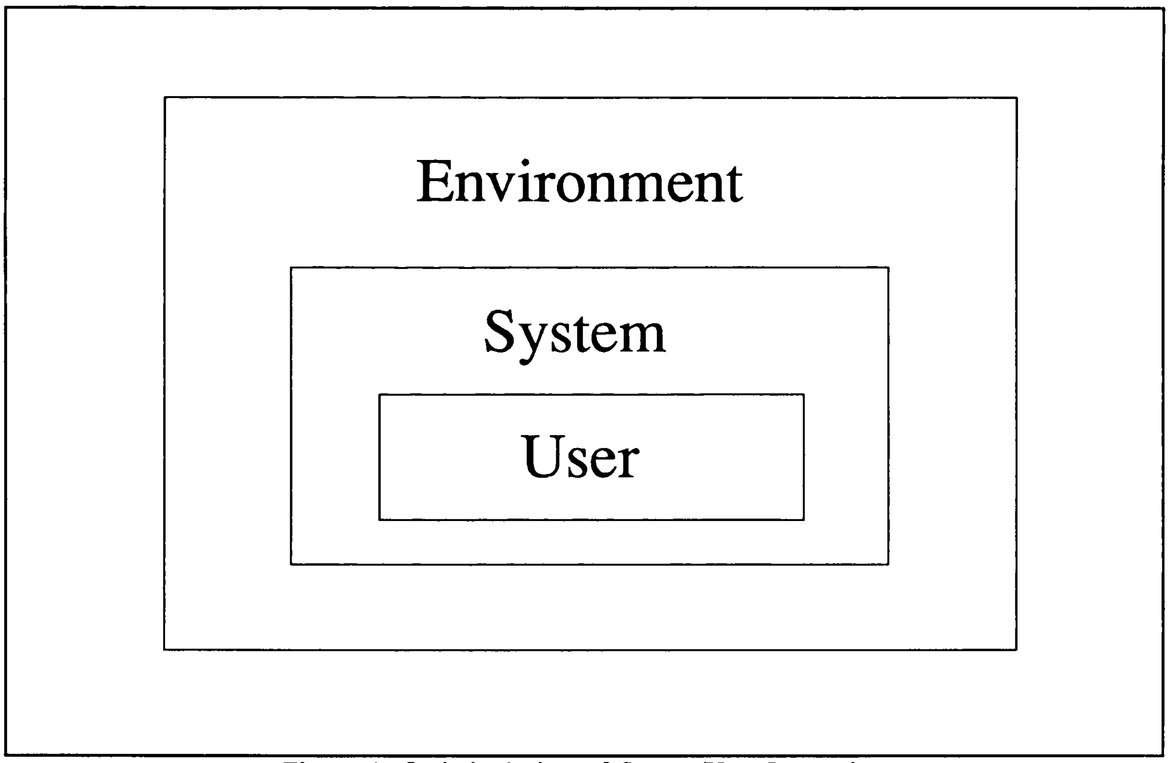


Figure 2: Optimised view of System/User Interaction within the Information Environment.

From this we can deduce that the human user is also part of the configuration and therefore a vital part of the Information Environment (see Figure 2). Once the user is modelled as part of the Information Environment, then interaction becomes facilitated with other users who are also working and represented within the same environment.

A truly multi-tasking and multi-user system is viable, and it becomes possible for all users to recognise the tasks that other users are performing, and so to monitor, to contribute and to cooperate for the benefit of all concerned.

Conversely, a user who is not correctly 'installed' in the Information Environment (i.e. one who has not been taught how to use the system, or who is assumed to be familiar with the more complex options) will not be able to operate competently or correctly, and can potentially make mistakes leading in the least case to inefficiencies, or in the worst case to damage of some kind.

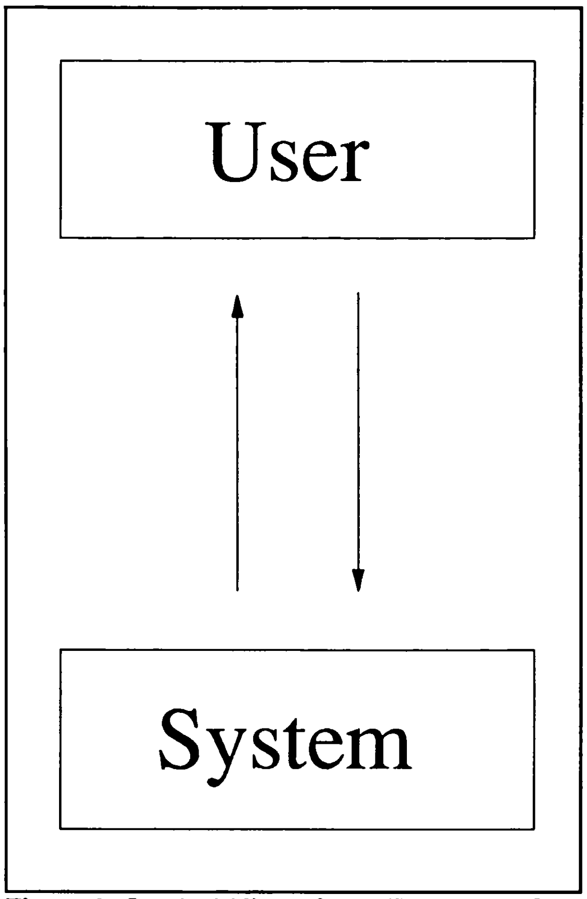


Figure 3: Standard View of User/System Interface.

The more typical practice of considering the user as being distinct and isolated from the rest of the information system (see Figure 3) means that any and all exchanges between user and system must pass through a single communications channel, which leads to significant problems concerning data supply and data reception for the user. However, once the users are recognised as central components within the information system, it is then much easier to establish more diversity in the nature and number of data communication channels, a development that suggests advantages for information handling.

As an example of why the integration of the user into the system benefits tasks being run by an application, it is clear that an advanced user will be able to take shortcuts in running the application, and will not need to use a 'help' facility except on rare occasions. The computer need not provide the help facility except when specifically requested to. Indeed, advanced users may find it frustrating to continually have a help prompt on screen, as this can significantly decrease the amount of useful screen area - the work area - available to them. Conversely, an inexperienced user will appreciate having an on-line help facility, and may prefer to be guided through every potential key press - which must inevitably reduce the speed of performing the task [Heppe 85]. The main danger to be avoided is the all-too-often occurring case of the on-line help being an exact duplicate of existing pages within the application manuals. This is rarely of use in itself, as it fails to supply genuinely relevant assistance[6].

Yet both types of users are essentially trying to perform the same task, and this means that the application processes and the system must reflect not only the task objectives, but also the circumstances in which the task is being performed. This is done by recognising the user's level of proficiency, and hence requires incorporation of the user characteristics into the system's operation.

Having shown the need for a complete description of the Information Environment, the problem of representation must now be examined. The concept of the electronic description is a tentative and exploratory step towards the aim, intended to allow the incorporation of all relevant components from the Information Environment (user, data, circumstances, application, and so on) into a single (possibly extensive) unit. Crucial to the proposed construction of the description is an underlying architecture that permits description of the various components in a meaningful fashion. However, it is clear that if the description requires a complex architecture in order to be implemented, then there must be some way of building up the description of the working environment and its applications in order to produce the basis of the application. A 'generator' is therefore required which reflects, and is specific to, the Information Environment that must be modelled to perform the activity. The design of a suitable architecture and a generator which implements it forms a major component in this report.

6: For example, the UNIX operating system will normally provide on-line copies of the documentation, but the actual number of worked examples is small.

Section 1.5: Summary.

There is a clear requirement for an enhancement of existing information processing systems, and the nature of such systems will remain the subject of on-going research. This thesis reports on the experimental results from a study of one possible approach based on geometry, which attempts to develop architectural mechanisms for enhanced problem representation, and thus offers a greater understanding of the undoubtedly complex issues and problems which must be managed. The geometrical concepts described are used to provide the basis of a prototype mechanism for studying how geometry may contribute to understanding problems, as well as being used for implementing a solution.

The prototype system itself is not intended to be anything more than a minimal implementation of the more primitive components. Instead, it is directed at providing a workable environment within which further development work can be carried out. This in turn would lead towards development of a more formalised way of thinking, and the development of a formal methodology to be used for solving problems that may benefit from geometrical analysis.

Section 1.6: Thesis Structure.

The structure of this thesis is as follows. In chapters 2 and 3, examples of current systems and current research are described. The examples have been selected for their direct relevance to the issues that are central to this thesis. The contribution of the examples is discussed in chapter 4, before moving on to consider in chapter 5 how each of the key attributes may be brought together to aid in specifying an enhancement to computer-based information systems.

The key attributes must be placed into a context, and this is provided by the Information Environment concept which is introduced in chapter 6. A suggested implementation of the Information Environment is given in chapter 7, along with details of how the system might be used in example applications. Actual implementation decisions made by the author are described in chapter 8, with reference to the demonstration system; before detailing the central facilities of the demonstration system in chapter 9. Chapter 10 summarises the results of the work, and suggests further lines of research and development that the author considers worthy of consideration.

Chapter 2.

Current Systems.

Section 2.1: Overview of Current Systems.

In this chapter, we are interested in considering the nature of current commercial products, and in identifying the aspects of those systems which are devoted to the general tasks of information administration, organisation and handling. We study how appropriate components that are fundamental to the system may be blended together in order to develop an enhanced information system architecture. This in turn will lead us to the suggested concept of the Information Environment. As implied in section 1.3, few current systems appear to be built upon the basis of an information architecture that could support such a proposal. Part of the advanced nature and suggested benefits of the Information Environment stems from the objective of structuring all components, at all levels. Therefore, it is relevant to consider the current systems which function at many different layers of operation and activity, to show how they have helped in identifying concepts used in the environment.

The normal sequence of events to be carried out during a complete information processing job involves a number of stages, starting with an identification of the primary goals of the job or 'application'. Next, tasks are defined that must be completed in order to attain the goals. Finally, the tasks are performed. The first two stages can be conveniently considered as a single group, as they are largely concerned with direction or policy in carrying out applications, while the latter stage or group is concerned with actual implementation of the policy. The first group is therefore that of the management packages, or activity analysis and management tools. These organise the specification of the information processing tasks which must be carried out, and so are used for coordinating the selection and use of the second group systems, the functional tools. The latter group - also known as application packages - have a clearly defined set of objectives to achieve.

We will proceed to discuss examples from each of the two groups, and show that one of the main reasons why the packages within the two groups may not be as widely used as might be hoped is because they often cooperate poorly, or have weak connections with the lower level tasks, or require independence from the higher level aspect of information processing. Thus the general cause of such failure seems to be due to an inherent inflexibility resulting from limited integration with other systems, and is suggested as an important obstacle to be overcome before information systems can make any further advances. However, it must be recognised that failures can also result from various other issues apart from technical inadequacies or internal design; for example insufficient or over-complex control mechanisms, or reduced speed, or increased cost. While important points, these latter cases of speed and cost fall outside the scope of this research.

As stated, the first group of systems comprises methodologies and packages for defining and performing information processing tasks that may be loosely categorised under the term 'administration purposes'. Such systems are less concerned with the low level tasks of handling raw data, but instead are intended to help with analysing and identifying key management strategies at the higher and middle levels of information processing. Within the higher level of administration, systems such as project planners or project management tools help in the identification and coordination of goals. At the lower level of administration, there are packages which help in specifying the actual tasks performed. These 'development' systems build upon the broad directions identified and organised by project planners, and enlarge upon the details of the tasks to be carried out. They help construct strategies for 'filling in the gaps' between the start and end of the tasks that must be performed to achieve the goals. Packages such as resource management or critical path analysis systems help in improving the efficiency of the information processing tasks, for example by isolating potential 'bottlenecks' which could slow down tasks and thereby affect productivity. Examples of these sort of packages include Pertmaster Advance from Abtex in the UK, and Viewpoint 3.0 from Computer Aided Management of California.

The second group of systems tackle specific application areas, with a general rule that any given package will focus upon basic information processing functions.

A few of the packages in this category seem to be quite extensive, and apparently have facilities for handling a variety of related or connected applications. As such, they can appear to be well integrated within themselves. However, it is sometimes the case that the sub-applications are special cases or re-interpretations of the original tasks. For example, Lotus 1-2-3[7] combines spreadsheet, graphic and database facilities. In practice, the database is implemented by providing functions which can search for and select from the correct 'cells' within a normal spreadsheet, while the graphic presentation facility is an alternative depiction of data values existing within the spreadsheet.

We will now consider the types of products that are currently available in the two groups of administration and application tools, showing how they process information at the various administrative levels, and culminate in the manipulation and presentation of data categories for the user. We are particularly interested in the first group of systems, because they are concerned with more abstract information handling, especially involving relationships and patterns in information. Accordingly, the systems incorporate higher-level processing aspects that depend upon information representation architectures, and it is these that we wish to identify in order to improve on existing structures.

Section 2.2: Management Systems.

Products in the general category of management systems come in various forms, for use according to the direction of management objectives. The first and highest level of objectives is that of overall planning, and is addressed by project management systems. Below that comes the more detailed concept of the resource management system, which considers the identification of tasks and resources required for the implementation of the strategy already established. A new form of this latter system is the software management system, which will also be detailed.

7: Lotus 1-2-3 is a trademark of Lotus Development Corp.

Section 2.2.1: Project Management Systems.

These systems are designed to organise and coordinate the structural inter-relationships of project or work tasks in order to achieve a specific objective. The most important elements are:

1) That a structured approach should be used in the identification of activities and resources necessary to perform the tasks.

2) That the activities and resource usage should be organised in the most efficient manner, typically by breaking large problems into smaller, more workable components.

3) That the progress should be monitored, and the actual usage of resources recorded.

4) That the progress made should be reviewed, with reallocation of priorities, tasks and resources as necessary.

Thus, as well as being interested in tasks, the relationships of tasks to each other are also relevant and important. Current project management systems are sometimes derived from database programs that have been enhanced in a number of specific ways. For example, they may be extended to include working-day and working-week calendars so that tasks can be placed in chronological relation to each other. During the analysis, each stage of a project has to be broken down into its component activities, which are in turn divided into tasks. These tasks should ideally have clear start and end points, used to identify when they begin and terminate. Concluding a specific activity means achieving a milestone. Scheduling the various activities and resources in relation to each other using the calendar systems means that critical tasks are performed in the quickest possible time, or at the lowest cost, with minimal problems. The end objectives are relevance, accuracy and efficiency. Analysis is carried out by using a variety of standard tools which work by identifying tasks and milestones. At each stage, the resources used and the costs of these components are recorded, and a number of charts in different styles can be constructed to depict the project as a whole. The two most common fundamental depictions are the Gantt chart (activity against time) and the Program Evaluation and Review Technique (PERT) chart, which represents interactions between the tasks. The Gantt chart is normally depicted as a bar chart showing tasks and their durations, and frequently the resources which were allocated to those tasks when they occurred. The PERT chart is particularly popular for implementation in Project Management System (PMS) software, as the resulting network diagrams are familiar concepts for the developers to work with.

Examples of PMS packages include 'Harvard Project Manager 3' available from Software Publishing in London, and 'MicroSoft Project 4' from MicroSoft Corporation. The Hoskyns Group has produced a PC based Information Systems management package called 'Bridge', which combines a development methodology with project planning facilities.

The sort of task to which a PMS may be better suited tends to be concerning the coordination of a long range or long term group effort, with clearly defined start and finish points. A simple example of their use might be planning the construction of a house, while a more complex example is that of constructing a large ship. A PMS helps by making planning and updating tasks easier than would be possible using ordinary paper-based systems. In effect, it processes information at a higher level of abstraction, with minimal reference to the potentially confusing underlying lower level tasks or data. Using computer technology also means that several different plans can be produced for varying parameters, so that comparisons may be made. As the project progresses, the tasks coordinated by the PMS can be updated to reflect the reality of how long they took, or how much they cost in terms of resources or time. This means that there is a requirement for a monitoring or a logging/auditing facility to be implemented.

More generally, a PMS will focus attention on the issues which have a significant influence on system development, in order to minimise error or inefficiency in the overall production process. The end result should be a project development environment that has had good planning, established recognisable progress and achievement goals, incorporated ample facility for refinement of plans, ensured coordination and consistency between all tasks within the project, and - importantly - can make use of prior project experience and use it to produce new experience, which can again be applied on later work.

The intrinsic paradox for most project management systems is that they have to be generalised in order to facilitate the more desirable goal of a less complex but more complete working environment, but at the same time they need to be specific to the management goal in order to achieve greater task efficiency and relevance.

Furthermore, providing the system with a suitable mechanism for processing all the tasks, resources and time components for projects means that the PMS packages must necessarily be of intricate architecture, yet their actual operation may conversely require the use of very straightforward control options, which insulate the users from the internal complexity. No matter how colourful or 'exciting' the interface is, it will often be built upon a rather fixed or inflexible option scheme. Such mechanisms can result in the users feeling that they have limited control over a nicely presented package, and as a result management users may be reluctant - or unable - to become sufficiently familiar with the system to use it to good effect. Certainly, a better understanding of a package would seem likely to produce more valuable results when the package is used, and the claim for the value of having a detailed understanding of tasks has been described as both instinctive and provable through direct observation during a study of Management Information Systems [Ein-Dor 86], although other sources, such as [Ackoff 67] are less convinced of this.

The goal of a Management Information System (MIS) is in assisting managers in their duties by helping with the collection, processing and presentation of information which is used in management tasks. Accordingly, MIS are contributory to the objectives of project management systems. The actual value of such systems can be very difficult to determine, owing to the varied nature of tasks and circumstances involved. One approach used for measuring value is by the use of tools such as 'business games', which are highly complex simulations of management situations, 'played' over an extended period. The Ein-Dor study used such a business game involving an MIS to test several hypotheses concerning the value of such systems, but in doing so lends itself to a number of important criticisms. Firstly, there is no discussion of whether the business game used was accidentally or intentionally geared to reflect the use of an MIS, and secondly, there was no test of the alternative hypothesis that use of an MIS could actually reduce the likelihood of success.

The general conclusions included the point that a better understanding of MIS function and applicability (termed 'familiarity' in the report) did indeed produce a significant correlation with success in the studied business game, especially at the higher levels of management, and made the statement that:

> 'greater use by top management of MIS increases the likelihood of success'.

When considered from a 'real-world' viewpoint, this is perhaps rather an unusual conclusion to reach. Given that the study was carried out some years ago, it would seem reasonable to suppose that such a positive result would have been rapidly reflected by increased usage of such products, but this does not seem to have been the case. It is worth considering whether in fact an MIS system used outside a controlled and bounded environment is in fact of any value. Certainly, there is little evidence to indicate popular use or even acceptance of such systems in the commercial world. The fact that significant numbers of managers do not use such systems (despite the general awareness of their availability) could even suggest that managers - who should know their jobs - may actually be correct not to use them.

Indeed, a survey carried out in 1988 indicated that the percentage of end-users requiring some form of MIS dropped from 4% in 1987 to 3% in 1988 [PCWeek 88]. Thus not only is the user-base demand for MIS comparatively small when compared with Information Retrieval/Reporting (23%), Application Development (16%) and DTP/Word Processing (24%), but it actually dropped in importance over the period of study.

Work carried out by Ackoff [Ackoff 67] suggests that few of the MIS that have been put into operation meet their expectations, and:

'some have been outright failures.'

He believes that these:

'near- and far-misses could have been avoided if certain false (and usually implicit) assumptions on which many such systems have been erected had not been made.'

Accordingly, the concern expressed in 1967 over the value of such systems appears to remain today, as borne out by quoted survey; consequently their actual worth must remain an unresolved point for the moment.

Section 2.2.2: Resource Management Systems.

As a contrast to Project Management Systems, Resource Management Systems (RMS) are rather easier to apply - possibly because they deal with the more 'concrete' problems found at the lower levels of information processing. They are concerned with the organisation of resources in order to achieve the specific task or end goal, once the tasks or goals have been ascertained by the PMS. Hence RMS form the next stage down in the managerial process, in that the end goal or task will have been defined, and correspondingly, the starting point(s) would be clearly recognised. The RMS is used in organising the execution of the tasks from start to completion, and for identifying the key points where enhanced efficiency or speed will improve the overall execution of the task.

The concept of RMS is one that is ultimately relevant to the implementation of an Information Environment - such as that described by the GENIE-M system proposed in section 8.2 - because in the larger model of the electronic Information Environment, there seems to be a requirement for a coordination of resources above and beyond the standard computer I/O resources. Any computer-based system that implements an

Information Environment will almost certainly have to coordinate the resources to be used and which have an 'existence' within the structure, and therefore the methods of operation used in an RMS will offer valuable data during the development of important components in the suggested information system architecture.

A typical RMS for application purposes is based upon the proven concept of Critical Path Analysis (CPA). All tasks comprise activities and resources, and the aim of CPA is to optimise the application of activities and resources to meet or improve upon timing, cost and quality measures.

Breaking down a task into its activities - a process started at the project management level - means that relationships or 'dependencies' between activities can be identified, although it should be emphasised that the accuracy and value of the identifications will depend heavily upon the analysis techniques used. The dependencies can be portrayed as a 'network' of activities, which can be made very complex by the fact that some activities can run concurrently with other activities, while others operate only on a serial basis.

A critical path is identified for a network such that, if one of the activities on that path were to be delayed, then the completion of the entire task would also be delayed. A Resource Management System can help to isolate the key activities on such a critical path, and so focus effort on improving those activities in some fashion. Furthermore, an RMS allows modifications to be made to the activities carried out - for example, by replacing old production tools with newer and faster ones (that might be more expensive) - and this enables the manager to examine ways in which the task efficiency may be improved, in the same way that the popular 'what if' approach has helped the growth of spreadsheet packages.

An example of a combined project and resource management system is the package 'Pertmaster Advance' from Pertmaster International of Leicester, for the IBM PC computer. It is operated by entering the activities, dependencies and resources of the task, as well as other parameters such as cost and timing constraints. The package can then produce a number of reports:

1) A standard report that details the earliest and latest start times to commence each activity.

2) A project and period chart to display overall task duration, and any critical path found.

3) A resource usage report.

Packages such as this combine tools for deciding upon information handling policy with the coordination of resources necessary for the information handling itself. Accordingly, such packages are relevant in establishing the higher-level aspects of a generalised architecture for an enhanced information system.

Section 2.2.3: Software Configuration Management.

We can now move more towards a consideration of computer system involvement as a complementary part of a management task, rather than merely as a planning tool for helping in its specification. This 'half-way' house concept involves the use of the computer to help in organising the execution of the computer tasks themselves, and good examples of the issues involved are categorised under the name 'Software Configuration Management' or SCM [Bersoff 81 & Buckle 82].

This concept concerns the controlling of changes which affect many components generated during the life cycle of a software product. The number of components can be very large - especially for complex projects - and so the efficient management of component evolution is of great importance in order to prevent the project from becoming at best unwieldy and at worst unmanageable. The configuration management process would normally be applied throughout the project at all stages from conception to completion. While there are a number of tools available to help with SCM, the Department of Trade and Industry STARTS guide [DTI 87] has a list of thirteen functional requirements that suitable tools would implement, and as yet none of the available tools meet all the requirements.

Section 2.3: Development Systems.

In general terms, we can define a Development System as being any procedure or technique which is concerned with the successful construction of a (possibly large) system. For our purposes in discussing computer-based information systems, we can further require that the produced system will involve the use of information and computer technology as major components. The process involves detailed analysis of the task or tasks to be carried out, and this in turn requires that the problem be broken down into clearly defined components, as described above in section 2.2. Quality control is important to ensure that a task specification is being met, and further that task execution is performed to the required standard.

One of the main ways in which the quality and accuracy can be (theoretically) improved is by the use of Software Engineering techniques, in conjunction with other tools such as program generators and 'Integrated Project Support Environments' (IPSE's). In this section, we will examine how these tools fit into the task environment, but as yet do not always integrate fully with each other or lower level tools. The purpose of this discussion is directed more towards the end goal of identifying issues that affect the design and specification of an enhanced information system architecture, although the consideration itself also helps with the development of a prototype system.

Section 2.3.1: Software Engineering.

In 1972, Terry Baker wrote a major article discussing the so-called 'software crisis', in which he brought together ideas on structured programming, top-down design and implementation, the chief programmer, the chief programmer team, and the documentation librarian [Baker 72]. This article provoked considerable research work on systems analysis, design and construction, and which has subsequently become identified as the area of Software Engineering. The topic has various definitions and meanings depending on the author being read, but generally the emphasis is upon promoting a discipline for the production of complex (and not-so-complex) software.

Software engineering takes the form of the appropriate use of a variety of techniques for dealing with software as a product that requires engineering quality precision. This means developing systems with regard to software specification, design, implementation, testing and maintenance. Concentrating on specific areas in a methodical manner should hopefully produce a much better development environment, and thus begin to ease the difficulties of developing a powerful system.

Despite its age, Software Engineering is still referred to as 'emergent' or 'new'. This is because although it has undoubted benefits, the techniques are not always sufficient in themselves to guarantee the successful conclusion of a development project. For example, there are still significant areas of influence which necessarily fall outside the software engineer's control. Such external aspects which are relevant include training, parallel hardware development, individual module and integration testing, as well as the very important matter of customer acceptance.

Despite certain limitations, there seems to be general agreement that a methodical approach to system development is preferable to a haphazard attack on the problems. Hence there is a very real need to learn from the benefits of Software Engineering principles, and to apply them in other applications, such as the design of user interfaces.

Evidence for this is observed in a number of references, including [Tagg 87]. In this reference, the author showed that Software Engineering approaches emphasise the System as being the object of central concern. But given that the user may be carrying out a number of tasks within a given environment, it is perhaps more important to consider the user as being at the centre of a number of systems. As Tagg points out:

> '[by adding the systems together] into a set of unified facilities for the end user ... he can use all of the computer services from a single workstation in a consistent way'.

Most texts on Software Engineering ultimately emphasise the 'waterfall' model for development, which has five basic phases often called the Software Lifecycle:

1) Specification.
2) Design.
3) Implementation.
4) Testing.
5) Maintenance.

Within each of these phases, a number of techniques, systems, methodologies or frameworks may be available. Some of them address more than one phase - with a corresponding danger of insufficient attention being given to the detailed or subtle problems possible at each stage - while other systems provide an in-depth philosophy for working in specific phases such as testing. Furthermore, many of the techniques tend to emphasise the earlier stages of the lifecycle. This is because, regardless of how well a program is structured and coded, it should be the result of the earlier design and specification phases, and so any errors or inefficiencies incorporated in those phases will reappear throughout the whole system. Some authors have carried out work which suggests that as much as 46% of the entire development effort is expended in the initial design and specification stages [Boehm 75]. That problems do occur later is made possible because of the inadequacies and imprecision of 'natural language' design techniques, and may even be inevitable given the dynamic and complex environments which are typical of large-scale software development activities.

As decisions are made with regard to the design of the software, there is a certain obligation to justify or prove the reason for each of those decisions. Some automated methodologies allow for the recording of each of the steps taken during the design and construction phases, and this data can assist with the evaluation, but again the difficulty is in the expression of this material [Maddison 83]. Work is being done in the development of frameworks [Coleman 88] that support the recording of development 'history', at the same time as reducing the danger of flooding the designer with too much or irrelevant background; and all while avoiding reliance upon specific Software Engineering methodologies.

The use of such a framework aids the communication and comprehension of design decisions relating to structure, component functionality, data representation and choice of (software) component implementations. However, the emphasis is still primarily upon the software issues, with reduced consideration of the user or hardware requirements or limitations. Even more notable is the absence from certain systems of any incorporation of systems behaviour principles, a particularly interesting omission in view of the possibility of behaviour contributing to the limitations of systems.

Generally, software design methodologies make use of a variety of graphical notations and regulations concerning the permitted or preferable design representations. A summary of these techniques appears in [Peters 80], but the existence of a large number of such techniques is evidenced by a survey in Holland which identified at least 24 techniques [Blank 83]. Perhaps the most well-known works on this area include deMarco [deMarco 78] and Gane and Sarson [Gane 79], with other important techniques including Yourdon's 'Structured Design' [Constantine 79], the Ministry of Defence 'Mascot' [MASCOT 83], and Jackson's JSP and JSD systems [Jackson 75 and Jackson 83].

Section 2.3.2: SSADM

One particular method of relevance and importance in Great Britain is the Government approved standard system, SSADM (Structured Systems Analysis and Design Method) [NCC 86 & Downs 88]. Its derivation is from the work of Learmouth and Burchett Management Systems (LBMS), who worked in cooperation with the UK government's Central Computer and Telecommunications Agency (CCTA). The requirement was that the methodology should be self-checking, use tried-and-tested techniques, be configurable to user needs, and simple to teach. The LBMS methodology was finally accepted in January 1981, and subsequently became a requisite for all government projects in early 1983. SSADM determines how a systems development venture can be governed, but with the potential for reconfiguration according to localised demands. The basic method is to reduce a project down into phases, which are subdivided into stages. Each stage is further divided into steps, which have a list of tasks, inputs and outputs.

Section 2.3.3: C.A.S.E.

A logical development of Software Engineering is to use computers themselves to assist in the coordination and presentation of the (often large) quantities of data being manipulated. A simplistic but workable definition of Computer Aided Software Engineering (C.A.S.E.) states that [Sommerville 89]:

> 'Computer Aided Software Engineering is the term used to refer to the support of the software engineer using software tools.'

C.A.S.E. is of particular importance when the objective ultimately or partly requires the production of a computer system. This is because much of the material will already be present within a computer, whereas most of the tools described in section 2.3 do not require that the end result of their usage must always be the production of a computer system, nor do they necessarily require a computer in order to operate. The specific problems that C.A.S.E. is supposed to target concern the design and generation of computer software.

As the power and memory capacity of computer hardware has improved, the complexity of software has similarly increased. Yet the task of software design is one that has largely remained disorganised, resource intensive and tediously slow. There have been several well-publicised examples of development projects that have failed to meet deadlines, including Lotus 1-2-3 version 3, which was finally released approximately two years behind schedule. The problem is that as the complexity of the project increases, so the requirements for development tools and good communication procedures increase. It is to problems of this nature that C.A.S.E. tools are addressed, but so far with limited success. Although C.A.S.E. tools have helped in certain specific areas, benefits can be quickly lost through trying to integrate the results with the remaining issues which are not within the scope of C.A.S.E.. Accordingly, the need for integration is increasingly quoted as a major thrust of development of such tools, ranging from considering the small scale concerns of individual stages within the development lifecycle, through to the full scale Integrated Project Support Environment.

A report on C.A.S.E. technology and its market and current products was issued by the research group Ovum in 1987 [Ovum 87]. The report defines C.A.S.E. products as falling into three categories. The first and highest level category is the true C.A.S.E. tool, or the closely-connected concept of the Integrated Project Support Environment, which is discussed in more detail in section 2.3.3. The second category or middle level consists of 'Analyst Workbenches', which are tools for specific analysis and design problems. A set of definitions in a data dictionary and an underlying design database are used in conjunction with presentation software to allow the developer to experiment with system diagrams. The third and lowest layer category comprises automatic code generators and Fourth Generation Languages (4GL). These tools generate most of the code required for an application by building upon some basic components such as database access modules. The remainder of the code is constructed out of the specification prepared by a programmer or systems analyst.

The Ovum report pointed out that in general, products developed for use within the latter two categories tended to be more effective when companies specialising in the categories collaborated with each other, rather than by attempting to develop techniques already perfected elsewhere. As an indication of this, the research report found signs of an increasing move towards integration or cooperation between analysis tools and design tools.

However, a more recent report on C.A.S.E. tools included in [Grindley 89] suggests that IT managers remain unconvinced of the value of such tools. Although they have been presented as the solution to system development retardation, 41% of all IT managers still have no plans to implement them. Of the 32% of IT managers who have used them, one in five subsequently rejected C.A.S.E. tools because of poor cost justification. According to this report, emphasis seems to be moving away from meeting project deadlines, and more towards integrating IT within corporate strategy.

Currently, the most popular C.A.S.E. tools seem to be Information Engineering Workbench, from Knowledgeware; Foundation, from Arthur Andersen; Maestro from

Softlab; and Case*, from Oracle[8]. However, IBM has recently announced its C.A.S.E. tool, called the IBM Application Development/Cycle. The key components of AD/Cycle are an integrated set of analysis, design and production tools based around a single application development control point called the Repository. This enables groups working on different parts of a development to share and exchange project information.

Section 2.3.4: I.P.S.E.

Integrated Project Support Environments (IPSE's) are usually multi-user systems that are aimed at managing software development projects that require large groups of analysts and programmers. They are derivatives of the need for an integrated set of C.A.S.E. tools, and seek to control every stage of the system development lifecycle with a major objective of supporting the development team in all the necessary work. The support is provided in part by maintaining a database of interdependent objects such as program modules and related documentation. Programmers are kept aware of latest versions of code for which their own work has some sort of connection.

Any modifications to documentation (such as changing a single data name) might normally take a long time and introduce potential problems of consistency. An IPSE would provide a cross-referencing facility to speed required changes with complete accuracy. Furthermore, the documentation can be automatically prepared to conform with the project standards by referencing a standard library which is defined at the start of a project. An IPSE might also be used for configuring the various program modules to form a complete application system, and this is a process called configuration management.

Section 2.3.5: Analysis and Design Tools.

Structured Analysis and Design methodologies typically use diagrams to show the relationships between the entities of the application, the processes manipulating the

8: It is interesting to note that Case* stands for Computer Aided *Systems* Engineering.

entities, and the flow of data between the processes. The validity of the resulting data model is judged by its ability to present the constraints or characteristics of the data needed. Coordinating the large amount of material produced by such methodologies requires a lot of effort in order to avoid problems of inaccuracies or inconsistency, particularly if the methodology is paper-based. Furthermore, it can be tempting to 'short-cut' the rules of a methodology, and this is even more tempting if mistakes have to be corrected later in the project.

However, personal computers allow the use of methodology packages (often called Workbenches) which replace the paper documentation with screen-and mouse-based manipulation tools. A difficulty arises with regard to the level of the workbench, in other words whether it merely offers diagramming tools for the data model construction, or whether the workbench can actually model all the required features. Certain facts cannot be represented meaningfully using a diagram, such as definitions or user responsibility.

> 'In order to be effective, ... (a) workbench needs to support both the diagrams and a background dictionary, or encyclopedia. Furthermore, there must be tight integration between the two. For example, when an entity is created using the diagrams, an entry should automatically be created in the background dictionary ready to be filled in with additional details - its abbreviated name, its definition, volume figures and so on. Ideally, the analyst should be able to swap from diagrams to encyclopedia during dialogue with the computer.'[Pearce 89]

Most workbenches are American in origin. An example is the Excelerator system[9], which includes a graphics facility for developing and updating system diagrams, and a dictionary for storing and cross-referencing design data.

9: Excelerator is a trademark of Index Technology.

Section 2.3.6: Code Generators and Fourth Generation Languages.

The main difference between automatic code generators and fourth generation languages (4GL) is that the former produces code for a third generation language which can be compiled, while the latter produces code which needs to be interpreted and is often less efficient. Another important difference is that the 4GL tools tend to be suitable for end-users, while code generators are orientated more towards the professional developer.

An example of a code generator package is the Application Productivity Systems Development Centre, which can run under DOS on a PC or on IBM mainframe computers running MVS or VM. As well as providing code generation facilities, it supports prototyping and the design of software modules. It includes a facility which enables the programmer to write code in a high-level language similar to COBOL, and this can then be automatically translated into genuine COBOL source code for CICS, IMS, DB2 and VSAM environments on IBM mainframes. Database code and common functions can be included by use of a macro facility. However, it does not handle the early stages in the logical design of a project, and so is intended to work with rather than replace the analysis and design tools.

Section 2.3.7: ISDOS.

The ISDOS system (Information System Design and Optimisation System) described in [Teichroew 77] was a system intended to aid information system development by establishing an automation process for taking user requirements (specified using a Problem Statement Language - PSL) and analysing the statements (using a Problem Statement Analyser - PSA) in order to produce a set of documents which includes the system analysts' formal specification. An optimiser was included in order to make generated code as efficient as possible. The aim of incorporating all aspects of systems work into a complete automated package proved over-ambitious.

Section 2.4: Data Handling Systems.

Data Handling Systems are specifically intended to work with particular kinds of data and produce a specific end result. In the majority of systems, they work explicitly under the guidance of the end user, who then has a larger amount of work to do in order to obtain the desired result. For our purposes, we consider two categories of Data Handling Systems: Standard Text Description systems, and the more recently defined Graphic Presentation Systems.

Section 2.4.1: Standard Text Description Systems.

In order to provide a greater degree of flexibility, while maintaining a minimal requirement for learning new systems, standard text description systems (STD's) work with a minimal amount of user support, relying instead upon the user being familiar with the main utilities provided on their system, specifically the text editor. This is then used to create files which have data and commands interspersed but distinct. This file is treated just the same as any ordinary file by the user's system, but once completed, it can then be processed by the STD.

Typical of these systems are the more dated text processing systems such as the UNIX based '`nroff`' or the more widely implemented TEX. Here, no real attempt is made to provide the user with a friendly presentation - instead, the user is expected to become familiar with the commands necessary to achieve the desired result. This has the advantage that each of the programs (editor and formatter) can be made distinct and so learned separately, but at the same time it reduces any possibility of integration of operation.

However, in order to give the system more capability, the complexity and range of commands increases to the point where it becomes difficult for a user to construct the desired document. It is then necessary to use a front-end package to help with the construction of the underlying document structure, and this approach has been used to enable the development of more flexible systems.

These systems include the Office Document Architecture (ODA) which goes to considerable lengths to define not only the structure of the commands possible within a conforming document, but also describes the actual structure of the document itself by providing amongst other things 'the definition of an abstract document model for presentation of office documents' which will 'facilitate the interchange of office documents' [ECMA 85]. ODA further provides for the definition of character and photographic content, but in order for this to be achieved the actual command and control information contained within an ODA-conformant document is considerable. [Brown 89] gives a good summary of ODA as a hierarchical and object-orientated model, where objects represent components of the document and attributes provide information about the objects.

A more recent but similar system is the Standard Generalised Markup Language (SGML). This is an International Standards Organisation (ISO) Draft International Standard [ISO SGML] for text processing, which is designed to enable authors to 'mark up' their documents on the basis of their logical structure rather than their final appearance. Abstract names, such as section and heading are used rather than specific formatting commands; since they can then be translated by the appropriate output device using a driver package that 'understands' the construction of an SGML document. This is clearly of benefit to any system working within an information environment, since it facilitates data exchange, although with limitations as to the range of data types which may be incorporated. [Brown 89] points out that, unlike ODA, SGML is a generic markup language rather than an object-orientated document description:

> 'This means that an SGML document, at its simplest, is a string of characters consisting of the text of the document interspersed with markup commands to identify the start and end of each logical item.'

By contrast, ODA documents have their structure description separated logically from the content of the document. With the latter systems, the greater flexibility naturally improves the potential for better quality documents, but the extra control information required means that it is entirely possible that the amount of markup descriptions for the document may actually exceed the quantity of material that forms the body of the text or diagrams.

Section 2.4.1.1: $\TeX$

The $\TeX$ [Knuth 86] system was developed by Professor Donald Knuth to deal with the problem of handling scientific and mathematical material. The coding scheme is public property, but its complexity is such that a dedicated package is required to use it, as an ordinary text editor would not be sufficient. Implementations are available on a variety of computers, including UNIX and the IBM PC machines. Its advantages encompass powerful text handling with minimal system requirements, and also portability across heterogeneous computer systems. As well as formatting equations, it is possible for authors to design their own fonts.

Section 2.4.2: Graphic Presentation Systems.

The variety of document production systems has helped increase the speed with which documents can be constructed, as well as greatly improving the possibilities for their appearance. Many of these documents have a basic objective of providing information to the reader, perhaps in the form of a company report document. However, once the document has been prepared, it is a common requirement to be able to present the material in a summarised form to an audience who can then read the detail of the actual report at their leisure. The typical circumstance of this would be the so-called 'Executive Summary and Presentation'. Accordingly there is a need for an information environment tool which can manipulate the same set of data in a variety of fashions, establishing one which is suitable for the specific needs of a given application.

Undoubtedly, a major factor influencing the success of such a presentation is the visual impact. Given that much of the data will be stored and prepared beforehand, and subsequently manipulated, probably using the same machine, it would make sense to use the preparation machine as a multi-purpose tool for constructing the visual presentation as well. Furthermore, the process of turning data into a satisfactory visual depiction would remain under the control of the data originators, and so is less likely to suffer from inaccuracies that could be introduced through misunderstanding.

Until approximately ten years ago, the only computers that could give reasonable visual presentation were the larger and more expensive dedicated workstations. The arrival of the desk top personal computer brought the promise of cheaper alternatives for graphics which are now being realised. The early personal computers had the capability to construct monochrome slides by printing graphic information onto overhead transparencies using a high-quality printer, but the problem of colour depiction was a major obstacle until the arrival of the cheaper-cost, high-performance graphic standards such as IBM's EGA (Enhanced Graphics Adaptor) and the more recent VGA (Video Graphics Array). These improved displays - and similar technology on other computers - provide the facility to construct clear and appealing displays which can be photographed and then projected for an audience.

However, there are few packages which allow for the appropriate configuration of all possible peripheral devices. This limits the ways in which computer systems can be configured. In other words, although a computer may be operational in the same room as a slide projector, a video unit, and a sound reproduction system (voice or music); there is no straightforward way in which the computer can be connected to all of the devices at the same time in a sensible and flexible fashion. A number of packages are directed to using a host computer to control a specific peripheral, such as an overhead projection system - a good example being the 'StoryBoard' program from IBM[10]. This allows the user to construct a series of story 'frames', which can then be projected for an audience.

There seems to be a clear requirement for developing systems that can use the computer host as a *controlling* mechanism, rather than centralising the processing tasks. Only then will it be possible to develop more flexible multimedia systems, as these normally require fast and dedicated processing of the varied media types. As an example of the problem, some packages provide basic visual presentation facilities

10: Storyboard is a trademark of IBM Corp.

already, such as Lotus 1-2-3. But in most cases the options available for the construction of the display are severely limited. An alternative would be 'professional' style graphic or art packages controlling dedicated graphics processors, but many users do not require the sometimes daunting range of options available in these systems, nor do they have the time to experiment with the package.

Accordingly, the need seems to be not just for painting or drawing programs, but instead for packages which understand the various kinds of data that users wish to present; and which provide a range of tools for constructing a visual presentation using the data with various peripheral devices. This also means that, for example, when charting information, a sensible set of defaults should be used in the construction of the image, and each of the components of the image can be easily identified and altered according to need and for maximum visual appeal within the context of the presentation device.

Section 2.5: Idea Processors.

When developing material, it is often recommended to have a working plan of how the content will be organised for presentation. The three basic components of the structure are of course Introduction, Main, and Summary/Conclusion, and within this general organisation, there can be many sub-components, each requiring careful thought in order to ensure that they fit neatly within the overall plan of the material. The use of computers for document construction through word-processing helps to speed up the construction process, but this same increase in speed also implies that the structure of a document can rapidly change and become confused.

An Idea Processor is simply defined as a package that helps organise blocks of information - typically text [Hershey 85]. They closely resemble word-processors in that they will often incorporate text editing facilities, but the emphasis is on the positioning of material within the structure of the document, rather than upon its visual appearance or content. Idea Processors tend to come in two basic forms: the indexing system and the outline organiser. The former work by helping with the management of large quantities of data which need to be arranged into some sort of structure before the main document construction begins. Outline organisers, on the other hand, work in a 'top-down' style to represent each of the blocks of information within the document [Foster 74]. The user should then be able to enter indiscriminate data from a variety of sources (thoughts, plans, ideas, etc.) which can supposedly be organised into the hierarchical structure offered by the Processor.

For most systems, there seems to be little consideration given to the way in which data might need to be modified in order to fit the structure imposed by the Idea Processor. Often, the ideas must be broken down or rearranged to fit the enforced hierarchical structure. There are, however, certain categories of data do not fit neatly into a hierarchical structure of any form, such as images or pictures.

However, once data has been entered into the Idea Processor, it can be reviewed in a variety of ways. The data blocks can be labelled with titles or headings, and can be relocated within the document structure to experiment with improvements in the style. Minimising the amount of information on display, with the aim of reducing confusion can be done by 'collapsing' the various blocks of data to show only their headings. This means that the structure of the data should be much more easily determined. Examples of Idea Processors include Brainstorm[11], ThinkTank[12], and PC-Outline[13].

More recently, products have been released which combine the structuring objectives of Idea Processors with more detailed data base concepts and indexing/cross-referencing facilities to provide much more comprehensive capabilities. These are called Personal Information Managers (PIM) and are intended to do for textual material what spreadsheets and databases have done for numbers and relationships: to allow users to enter, retrieve, analyse and cross-reference potentially arbitrary pieces of data at will.

11: Brainstorm is a trademark of Caxton.

12: ThinkTank is a trademark of Living Videotext.

13: PC-Outline is a trademark of Brown Bag Software.

A typical design goal of the PIM packages is to help you access data in an orderly fashion, and further to help identify relationships that might not otherwise be noticed. Two major PIM products in this category are Grandview[14], and Agenda[15]. The former package is directed slightly more towards an outlining style, while the latter package is based more upon a database cross-referencing facility.

Section 2.6: Data Flow Control.

In the micro computer world, the main operating systems are DOS, which is fundamentally a single-user, single-tasking operating system; and the Apple Macintosh Operating System (although UNIX and OS/2 are also contenders). The latter system was designed from the outset to incorporate strategies to facilitate data exchange between applications, and at the launch of the Macintosh computer, a number of tools were provided that - amongst other things - enabled the user to 'cut-and-paste' data between different packages.

When the Microsoft Corporation announced the 'Windows' graphics environment for the IBM-compatible PC, the intended strategy was to provide a facility whereby limited multi-tasking in the PC environment would be enabled, and also to support data exchange between programs. This is achieved in part by allowing applications running under the windows system to exchange data in an agreed form, called Dynamic Data Exchange (DDE) [Schifreen 88]. This is a protocol providing for the exchange of data between applications running under Windows. It works because Windows is essentially a message-passing operating system. This means that any event which occurs within the system - such as a key-press or mouse movement - causes a message to be generated. This message describes what happened, why, which application was involved, and various other information. A 'well-behaved' application running under Windows simply reacts to the messages that it receives. Messages can be generated by other applications, as well as the Windows kernel itself. It is quite permissible for

14: Grandview is a trademark of Symantec.

15: Agenda is a trademark of Lotus Development Corporation.

an application to send itself messages in order to invoke additional functions. Messages can be sent to other applications, and these messages form the basis of the DDE concept.

A number of data exchange formats are understood by applications supporting DDE. Some of these are specific to PC applications, including Lotus 1-2-3, but others are more general. Amongst the general formats are:

1) Plain ASCII text, with a standard Line-Feed Carriage-Return sequence to delimit each line.

2) Bit image data in the form of a series of binary information representing blocks of pixels.

3) A metafile, which is similar to a PostScript file in that it consists of a series of instructions which describe the way the object should appear.

4) A comma-separated variable format, where each record is separated by comma, allowing the record to be of variable rather than fixed length. If the length were to be fixed, then padding or truncation might be necessary.

The format for interchange of information among remote heterogeneous components is a much more complex issue. However, there are a number of standards which are directed towards this problem, including ODA (Office Document Architecture), which is the ISO standard for a logical document structure used primarily for information interchange [ISO 84].

Section 2.7: Discussion of Current Systems.

The examples of current systems can be summarised as follows: The management systems attempt to analyse a problem. The development systems are aimed at a similar objective, but are concerned more with the coordination of resources used for solving problems, and specifying the use of applications and packages at a lower level. In effect, they produce a more formal specification of the problem. The Software Engineering systems - C.A.S.E. tools and I.P.S.E. workbenches - all highlight the problems of coordinating information handling, even within the specific environment of software development.

The difficulty is that at most of the stages, any given system works to produce a specific result, at which point its work may be considered complete. However, in some cases, the results may be applied to other (possibly lower level) systems, but the results may not always be sufficient, adequate, or even in the correct form for the lower level application. In other words, the higher level or administrative systems produce results that may be of use only within a limited scope. Typical of the high-level results would be a detailed diagram showing how resources and tasks inter-relate in order to achieve the desired corporate objectives. Yet the diagrams cannot be used by other systems outside the original generation package, they must first be understood by a human user, who then uses the knowledge to instruct the lower-level systems. In a similar way, the middle-level systems such as the Analysis and Design packages produce specifications which may well reflect the desired tasks, and yet only 25 to 30 percent of specifications generated by these systems are actually used and submitted to the lower level program specifiers or application generators [Gibson 89]. Thus, despite the apparent advantages of using C.A.S.E. and similar tools, there would seem to be limitations regarding the actual exportation and applicability of the results obtained from the higher- and middle-level layers.

The categories of current systems described are important, because they are each closely involved with providing for the handling of data in some fashion. These approaches to the manipulation of data leads to two important observations that are pertinent to the development of an enhanced architecture:

1) It may be necessary to design a better information handling architecture that offers more supportive and valuable information processing.

2) That powerful data handling on its own may not be sufficient for an enhanced information system architecture; there could be additional requirements for incorporating higher-level 'abstract' issues relevant to the processing of data, such as administration, control; and identification and support for data relationships and attributes.

Both of these observations form the substance for exploration in section 5.3 of this thesis, as an exploratory architecture for enhancing information systems - and the information environment derivative - are discussed. In order to develop an architecture, it is considered vital to review the work that has been carried out to date, in order to assess what has been learnt. Without this review, there is a danger that errors or misconceptions introduced during the development of the other systems may not be recognised and removed, resulting in less reliable systems of reduced benefit.

Chapter 3.

Current Research.

Section 3.1: Overview of Current Research.

The research areas considered here do not necessarily have the primary background of trying to address the topic of general information systems. However, in their direction and treatment of subjects, they provide evidence in the form of many pointers identifying the components contributing to information handling. When grouped together, the clues suggest reasons why an Information Environment is desirable and how it might be designed. Further, this leads to the suggested requirement for a suitable generator package to be specified.

As indicated previously, a major factor in an information system is the vital contribution of each user as part of the working environment. In order to function more effectively, the user must be presented with as much data as is minimally desired for information transfer to be successfully accomplished, and for adequate - as defined by the user - interaction to be facilitated. In order to make this feasible, the maximum possible range of data types and presentation styles must be supportable within the environment. The presentation styles used must necessarily reflect the data and data relationships, and since the user directly influences these aspects, the user's activities and preferences must be taken into account.

Accordingly, we must examine the general nature of information system methodologies, before moving on to consider those areas of current research work which are specifically directed towards the handling of many data types and relationships, as well as the corresponding variety of methods used for the presentation of the work: multimedia and hypertext systems.

Multimedia systems address the problems of handling potentially large groups of data which are formed from a number of varying media types. This objective introduces significant difficulties, not just for the real-time manipulation of data, but also regarding the establishment of suitable mechanisms for the communication and presentation of the data.

By way of contrast, in hypertext systems a structure is already imposed on the data. Text fragments are embedded in a directed graph with labelled edges and reference mechanisms are supported, allowing the user to traverse those edges. hypertext system user-interfaces are built using state-of-the-art technology such as high resolution graphic displays, sophisticated pointing devices, and Icons for command options. However, these systems typically lack a database-derived content query facility. Elementary string search facilities may be supported.

Following these two major research areas, we briefly appraise more general research components, before looking at the architecture of computer systems with a view to seeing how the techniques of information handling are currently being facilitated.

The conclusions gained from considering these research areas will lead to a better understanding of the derivation of the components of the Information Environment concept and its benefits to information system users. However, we must begin by considering the nature of methodologies which may be used for developing and organising information systems.

Fitzgerald et al [Fitzgerald 85] carried out a survey analysing the features of contemporary information system methodologies, describing a framework for the evaluation and comparison of such methodologies, and using it to analyse seven examples. They considered the background, philosophy, assumptions, scope and skills required, as well as considering the analysis and design phases of each.

They emphasised that the methodologies were intended for use during the development of an information system, and consequently focus upon such problems as delivery failures, poor resource usage or maintainability, and failures to meet overall objectives. Many methodologies solve large and complex problems by breaking them down into smaller, more manageable units, which could be tasks, problem areas, or even simpler but complete information sub-systems. As a result, the methodologies that Fitzgerald et al considered have a close relationship with the various phases of the software development cycle. In practice, the analysis performed by the studied methodologies was directed more towards the early development stages, particularly those of problem analysis and logical design.

The analysis of the methodologies was broken down into five categories:

1) The scope of the methodology, to establish which system development phases it dealt with.

2) The objectives and deliverables that would be achieved or obtained through the use of the methodology.

3) The philosophy underlying the methodology.

4) The explicit and implicit assumptions made by the authors of the methodology.

5) The hardware and software support (if any) for the methodology.

The Fitzgerald paper concluded that methodologies appear to be enlarging the scope of activities covered, and that organisations would increasingly use them as a standard for information system development. While welcoming such a trend, the authors pointed out that:

> 'there is still a long way to go to make it effective. We do not foresee any one methodology becoming dominant in the immediate future, and we expect more organisations to take one of the many methodologies available and tailor it by use to their particular needs and preferences.'

Many of the methodologies made certain assumptions, including the implicit requirement that information can be modelled, that models used by the methodologies for their tasks are adequate, and that extrapolation of those models is meaningful and 'easy'. Fitzgerald et al felt that there was:

'a need for more empirical studies to examine how methodologies are used in practice and to determine how effective they are.'

In another paper, Galliers and Land [Galliers 87] consider it important to extend the viewpoint used to study Information Systems, in order to incorporate:

'behavioral and organisational considerations'.

This is explained by a wish:

'to improve the effectiveness of Information System implementation *in* organisations and to assess that impact *on* individuals or organisations.'

Galliers and Land point out that if the research work carried out in 'the real world' has little applicability, then there is no relevance to carrying out the work itself. Accordingly, the research must take into account the nature of the subject and the complexity of the real world. These issues are of course fundamental to all kinds of research, not limited to Information Systems study, but nevertheless they highlight the importance of using the correct approach in our efforts.

In the research described in this thesis, we are interested in the modelling of information, such that a model used for a particular information handling activity should be as effective as possible. In order to understand this, we must examine the current techniques for simple to complex data handling, before considering the use of structures for larger-scale modelling of data groups.

Section 3.2: Multimedia Systems.

Where potentially large quantities of possibly widely varying data types are to be handled as a whole, then there is the requirement for correspondingly complex data handlers. A typical approach to these issues is categorised under the area of multimedia Systems. Generally, research involving multimedia systems falls under the wider category of study into multimedia communications, and is defined [Aceves 85] as that which:

'refers to the representation and dissemination of machine processable information expressed in multiple media.'

This means that with the increasing power of appropriate technology, the familiar tasks of data collection, processing and distribution have now become increasingly dependent upon computers. In fact, computers are considered as an essential tool which must be used for these processing tasks. As Aceves et al point out, the use of improved computer technology:

'...will permit user applications that require [simultaneous] processing and distribution of information in several media, including alphanumeric data, text, voice, computer graphics, and images.'

In fact, for reasons in section 7.2, it seems reasonable to suggest that this list should also include less obvious data types, such as environmental and attribute data, for example temperature or brightness.

Since computer systems are often considered to be a prerequisite for there to be any hope of achieving successful multimedia document systems, a natural trend of the current research is towards using the computer system to assist actively in all aspects of the data collation, manipulation and presentation. This idea has been proposed in [Negroponte 76], with the suggestion - or caveat - that:

'electronic displays (should) adaptively display not only what materials readers wish to see, but also how they would like it presented.'

The various features of current multimedia information systems can be evaluated to help with the task of identifying the minimal facilities that must be supported in order for the multimedia system to be classified as such, and in order for it to offer the maximum benefit to users. The main requisite features concern the interconnection of data; support for retrieval, editing and presentation; and maintaining historical integrity.

The first of these topics concerns the establishing of links between (potentially) related topics or ideas. Taken in conjunction with the problems of presentation and 'location depiction', the associated research area of hypertext systems is identified. A great deal of importance is attached to the objective of associating varying data types, and further being able to integrate them into a single conceptual document.

Establishing the mechanisms for control and presentation of multimedia data is work of considerable interest to current researchers at present. A number of systems have been designed which address this particular issue, taking into account the variety of peripheral presentation devices that may (or may not!) be attached to the processing host machine. In certain projects, specific control devices have been developed in conjunction with work done by human-computer interaction specialists. A good example of this is the small 'piano-like' keyboard used on the NLS/Augment system described in section 3.3.

If presentation of the data and inter-connections is of clear importance, then the mechanisms offered for modifying the data and the inter-connections must be at least as versatile. The need for flexible editing tools is paramount in order to assist with the rapid but (hopefully) improving status of a multimedia document. As data is modified either by the user or by external events, so the structure used for organising or categorising all the data within the multimedia system must also be updated. This updating process continually poses the threat of a confused resultant data structure. The matter is complicated by the fact that hierarchically organised tree structures are unable to represent plainly all the potential data relationships that may be required during a given information processing task. The user's editing tools cannot realistically be permitted to focus exclusively on the single job of data modification, as this makes no allowance for the effects of alterations upon the potentially complex data interrelationships, and so data alteration alone is a coarse strategy that inevitably promotes disorder by overlooking the context of the data. There is a need for the implications of such modifications upon structure to be incorporated in the editing tasks - and accordingly recognition of the data context must be facilitated.

The issues of retrieving relevant data from within a storage structure are studied primarily by research into database concepts, but there is an additional and wider-ranging research area of information browsing. This topic will be addressed in more detail in section 3.5, but is concerned in part with incorporating techniques of accessing the specific data from general data sets. The final problem of historical integrity is considerable. On the one hand, it is desirable to ensure that a given document be as up-to-date as possible, yet it is easy to envisage circumstances when it is desirable or essential to recover previous or even original versions, or to 'fix' the current version of the document so that it cannot be altered. Such requirements inevitably influence the nature of the control and manipulation mechanisms.

Section 3.2.1: Example Systems.

One project that emphasises the integration of multiple data types into a single document object is the Diamond Multimedia Document system [Forsdick 82 & Thomas 85], which is:

'a computer-based system for creating, editing, transmitting, printing and managing multimedia documents.'

Diamond works with five basic media types: Text, Graphics, Images, Voice and Spreadsheets. Each of these exist as 'objects' in a Diamond document, and can be thought of as having a 'physical presence' within the document - thus permitting a variety of editing and manipulative operations [Diamond 85].

The text and graphic data types are the familiar kinds, while the image, voice and spreadsheet types require a little more explanation. An image object is a digitised picture, a voice object is a digitised voice passage encoded by a vocoder, and a spreadsheet is a conventional tabular array containing numbers, labels, dates and formulae. When a voice object is to be depicted on a workstation that does not support a voice object peripheral, a small icon (a loudspeaker) is used to indicate that a voice object would appear at this point. The document author is able to associate a small text caption with this icon, that describes the content of the spoken passage.

Despite the general success of the techniques used in the Diamond system, the developers felt that there was still significant room for further work. Diamond permitted the construction and exchange of some extremely complicated multimedia messages, but:

> '...even with this greatly improved means for communication, there is still an aspect of human communication missing because of the static nature of documents: When interaction between two or more people over the contents of a multimedia document must occur, the relatively long time ... needed to send a document and receive a reply impedes progress. It soon becomes obvious that engaging in a real-time conference over multimedia objects adds perhaps another important dimension to the process of communicating.' [Forsdick 85]

In other words, there is a clear requirement that data processing systems being used for data communication (presentation) must take into account the continually changing state of the real-world, and furthermore that the changes should be recognised and incorporated in real-time.

As part of the work towards development of real-time multimedia systems, attention must be paid towards the problem of storage of the data. A suitable system must offer flexible document retrieval of potentially very large data groups, and almost certainly there will be a need for the storage to function within a distributed environment.

One of the systems to address this area is the Muse filing system [Gibbs 87]. Its objectives include providing basic filing operations, a distributed architecture, very large storage capacity, support for optical disk technology, content-based document retrieval and standard encoding of document information. Muse works with the same five basic data types as described for the Diamond system.

Of particular importance in the construction of Muse is the concept of the 'document servers'. These are processes which provide basic filing and query operations on documents - furthermore, servers can be added or removed without affecting other servers. In order to achieve this capability, certain restrictions were imposed upon the server's operation:

> 'the storage managed by one document server does not overlap that of any other server; the storage managed by a particular document server resides on a single network node (but a node may support more than one server); and a server itself does not communicate with the other document servers, so it need not be aware that they exist'.

While making implementation somewhat easier for the Muse system, such limitations nevertheless have repercussions concerning the final flexibility of the overall architecture. In order to control the operation of the servers, user requests have to be communicated by the use of processes called 'clients'. The clients encode all the server requests and replies for communication through the server interface. This interface is in fact the only common ground between server and client - the actual execution of either the client or the server is of no relevance to the other. The advantages of this client/server-based architecture are clear:

> 'Performance appears to be adequate ... the modular design and the separation of the clients from servers allow easy experimentation with the system'.

The Pleiade System [Nanard 87] is intended to be a professional quality document manipulation system, that interactively manipulates structured documents that include text, graphics, formulae, tables, and any kind of structured objects defined by the user. It incorporates an integrated editor/formatter that offers a 'nearly-WYSIWYG' display on the screen. The architecture assumes a structured description of the document, and utilises user-defined logical types to specify the formatting rules. Its approach is primarily declarative (rule-based), although it uses an 'iconic language' and object-orientated approach to operate directly on the visual entities depicted on the screen.

The underlying structures for a document consist of three parts. The logical structure in Pleiade:

> '.. is a set of independent trees, one per page area. Logical types decorate [format] the nodes of the trees. The leaves of the trees are the elementary components (tokens) of the document.'

This logical structure is used to express how the document components are hierarchically ordered, and provides the framework on which the formatting rules are applied. A page structure is used to represent the macroscopic geometrical aspect of the page. Areas on the page are used as containers into which document data is poured, similar to that employed by the Interscript system [Ayers 84]. Finally, a geometric structure is 'induced' by the positional rules associated with logical components, using the box model offered by Knuth [Knuth 79].

The system developers recognise the importance of the underlying document structure with regard to the system, although they are aware that the abstractions involved are perhaps more advanced than most users are currently acquainted with:

> 'We take up the wager [sic] that abstractions such as structuration, logical types and declarative approach can be accepted and used efficiently by common users, as far as they will be presented in a user friendly way.'

However, they recognise that the Pleiade architecture is such that the structure used for working with a document is a consequence of the operation of Pleiade, and does not represent a model of the actual document itself.

Other multimedia systems include MINOS [Christodoulakis 86], which is an object-orientated system; and CCWS, developed for military command and control activities [Poggio 85].

Section 3.3: Hypertext.

It is only in the last one or two years that hypertext systems have begun to offer commercially successful real-world applications, despite the fact that their conceptual origins date from the very earliest days of computers. Writing in 1945, the author Vannevar Bush proposed an electronic system called Memex, which would have resource control over unlimited amounts of data [Bush 45]. The idea was that it should be able to find any piece of data almost instantly, and further that it should be able to make connections to any other data item.

It was not until 1965 that Ted Nelson coined the term 'hypertext', subsequently defining it as:

> 'a combination of natural language text with the computer's capacity for interactive branching, or dynamic display ... of a nonlinear text ... which cannot be printed conveniently on a conventional page.' [Nelson 67]

In 1968 Douglas Engelbart [Engelbart 68] demonstrated a graphical workstation system called NLS (oN Line System). This could change data context both between documents and through different levels within the same document - thus depicting the basics of hypertext operation. In order to achieve this functionality, NLS used a hierarchical structure which allowed reference links to be established between levels and between files.

If multimedia research is directed towards the handling of varying data types and their subsequent storage and retrieval, then research into hypertext (also known as nonlinear text) is perhaps directed towards the representation and development of paths, links and annotations for the body of data - albeit of limited type.

However, despite the considerable effort being directed at hypertext systems, there is still a considerable paucity of applications that require and benefit from the use of hypertext as an information processing architecture. A good overview of hypertext systems appears in [Smith 88], while another more recent review of hypertext technology is contained in [Ritchie 89]. Here, the author points out that in fact, the use of non-linear structures such as that epitomising hypertext documents is not new. For example:

> 'Medical textbooks such as *Gray's Anatomy* (first published in 1858) rely heavily on cross-references including links between text and diagrams, text and text, and diagrams and text.'

Hypertext systems are generally very straightforward - a requirement that is justified by the desire to make the systems easy to use for the non-expert. Typically, presentations on a screen are closely connected with the internal storage of data or objects within a database. Links connecting the objects are provided as pointers within the database itself, and sometimes depicted graphically on the screen to help the user with control and selection tasks. Hypertext systems have therefore been described as 'networks', where nodes store the data, and links exist between the nodes. The reasoning behind the concept is that not all data which forms information can be represented or stored in traditional database architectures - for example large-scale text storage of library material, where cross-referencing (but not word-by-word indexing) is required. The linkages which can be established and used within a hypertext system represent the more important aspects of the system, as it is these which facilitate the nonlinear organisation and retrieval of material. The importance of links within hypertext is strong evidence of the requirement to provide relationship as well as data storage within a generalised information processing system.

Many hypertext systems operate with the assumption that the blocks of material to be displayed on the screen must have a one-to-one correspondence with the data nodes stored in the internal system. This is perhaps one of the more notable limitations on the implementation - and more importantly, visualisation - of hypertext systems. Additionally, there is the possibility that the suggestive emphasis of the word hyper<u>text</u> might limit the ways in which a user perceives or visualises the possibilities of working with the application.

The presentation of hypertext appears to have much in common with ordinary Windowing systems, but the difference is in the relationship with the underlying database - in fact, there is no requirement for any database in a genuine Windowing system (perhaps illustrating why the use of windows with hypertext or as a description of hypertext is misleading). This is because windowing systems have the primary goal of partitioning and supporting tasks, and are not normally concerned with the data being manipulated by the tasks. Similarly, existing database architectures on their own are not really suitable to qualify as hypertext systems, since the latter requires considerably more from the linking and user-interfacing facilities than is provided in even the best of database implementation or interrogation architectures: typically the database linkages will be based on inquiry using specific frames in rigid formats, and user interaction is provided by rigidly defined text-orientated command lines. The outline processors described in section 2.5 do not support any form of cross-referencing between material - although more recent versions are moving towards this feature, and hence become closer to implementing the hypertext concept.

A list of the main definitive features supported by a hypertext system could be as follows [Conklin 87]:

1) A database with a network of textual (possibly graphical) nodes, forming the hyperdocument.

2) Windows on the screen corresponding to nodes within the database. Names or titles are always displayed, but only a limited number of nodes (windows) are open at any one time.

3) Standard windowing system operators are permitted, including repositioning, sizing, and expansion from or reduction to icons.

4) Link icons are portrayed within the open window to show links to other nodes within the database.

5) Additional nodes and links can be easily created by the user. This may be to annotate, comment or elaborate.

6) Database interrogation can be by three methods:
   a) Following links and opening the corresponding windows for the located nodes.
   b) Searching the network or a subset for a specific string.
   c) Navigating the database using a browser that displays the network in a graphical form.

This latter facility illustrates a major aspect of hypertext usage, in that the nonlinear nature of the text, coupled with the on-screen representation of nodes, means that it is surprisingly easy for the user to lose the 'context' of where they are within the network. This problem is aggravated if the context changes as a result of indirect commands, rather than at the explicit direction of the user.

An example of this might occur during a search operation, where the appearance of the located item is the result of the user's command. However, the actual 'jump' of context in order to permit the item to be presented is an implied side-effect that causes the user to lose track of the stages of context change.

Another example concerns the 'Doomsday Project', which combined CD-ROM technology with a database of facts and figures about the United Kingdom. The CD-ROM system can be used to depict appropriate images, but it is comparatively easy to 'get lost' within the system, necessitating a back-tracking process to a known 'meeting point'.

The use of a browser or 'graphical view finder' should offer help by the portrayal of the interconnections in the database, coupled with a display of the exact route used to reach the current context. These finders could operate using principles which have been well established by researchers investigating structure editors and hierarchy editors [Allen 81, Furuta 89, Kimura 86 & Pitman 85].

However, all of the above description has been directed at the implementation of hypertext systems, and it is important not to neglect the contribution of such systems as a powerful mechanism for establishing and developing ideas. Traditional recording mechanisms (such as pen and paper, or conventional text or word processors) lead to 'flat' text.

This means that any exceptions to the linear flow of the idea have to be indicated with artificial digression signs such as parentheses (like this) or footnotes[16]. A hypertext architecture, by contrast, not only provides for but encourages such literary 'asides' which are of equal priority and status. The 'scattered' nature of data using networks supported by hypertext is thought by some researchers to be much closer to the normal human mechanism for associating ideas and concepts, resulting in a much more realistic and natural way of working.

Section 3.3.1: Example Systems.

A limited form of a hypertext system is that of the BLEND Experimental Electronic Journal Programme [Maude 85]. This project is less presentation-orientated than most hypertext systems, and studies the user requirements for electronic journals. The results are used to develop prototype software that incorporates the newly derived features. The BLEND program functions through an ordinary computer monitor, without assuming high-resolution graphics or even mouse control. As such, there are limitations on both the quantity and range of material available on screen at any one time.

16: Like this.

Development of the system began with a study of the approaches used by scientists to evaluate whether papers in journals were considered interesting to read. Three main techniques were found, all of which were considered inappropriate for ordinary linear display of text on computer monitors:

1) A filtering process, looking at the Title, Abstract, Results/Conclusions, References and finally the main text.

2) A preliminary filter of Title and Abstract only, then a decision is made as to whether the entire article is read.

3) Skimming through the entire article for new ideas without detailed consideration of content.

With these techniques in mind, the BLEND system was specified which structures text by splitting it into 'entries'. These can be of any length, but are normally restricted to single paragraphs or figures which can be depicted in their entirety on a single screen page. For any given document being displayed, the reader is allowed to step forwards or backwards through the text, or to jump to previously marked or searched-for locations.

These facilities on their own would make BLEND little more than a text-browsing system. Accordingly, the system has extensions to allow the addition of comments relating to specific parts of a document. Every entry in a document can have a comment 'tied' to it which may be read upon request. Furthermore, the comments can be read independently of the paper itself, and in fact accessed in the same manner as the paper.

Perhaps the most well-known hypertext system is the HyperCard system for the Apple Macintosh computers. Although it is a commercial product, it remains the focus of much research and experimentation in related fields. Its fame is primarily the result of being easily available, since it is 'bundled' as standard with all Macintosh computers. The official description of this package is:

> 'HyperCard is a personal tool kit that gives users the power to use, customise, and create new information using ... text, graphics, video, music and animation. In addition, it offers an easy-to-use English language-based scripting language[17] that gives users and opportunity to write their own programs.'

Furthermore, the official policy from Apple is that HyperCard is intended for use as system software, and as such it provides services for use within application software, rather than being a complete turn-key or end-user system in its own right. In practice, however, HyperCard exhibits many of the basic characteristics of a true hypertext system, although it has certain limitations which make it less than a 'pure' implementation.

The user's view of HyperCard is that of a collection of cards (342 pixels high, and 512 pixels wide - the same size as the Macintosh screen), which can be grouped into a file called a Stack. A single stack can hold theoretically 16 million objects, not exceeding a total size of 0.5 GBytes. Cards can hold text and/or graphics, and furthermore information can be shared between cards within the same stack. On a card, text (up to 30,000 characters) is stored in fields.

Actions can be implemented by the use of 'button' fields. Using the Macintosh mouse to 'click' on a button causes a specific event to occur, depending on the purpose of the button. The cards can also have a common background image (which is shared between all cards in the stack), or individual images which belong to a single card only. There are five basic levels of accessing a HyperCard stack:

1) Browsing: A limited set of menus are displayed, searching is permitted, but data entry or modification is not.

2) Typing: New data can be entered, or existing data altered.

17: This language is called HyperTalk.

3) Painting: Graphic images for cards can be created and edited.

4) Authoring: Buttons and fields can be created and altered. 'Hot-keys' can be defined[18].

5) Scripting: HyperTalk scripts can be created or edited.

HyperTalk scripts work using events - when one occurs, the appropriate script is executed. HyperTalk has aspects of a simple object-orientated language. Examples of events include pressing certain keys, using the mouse, or even recognising the inactivity of significant idle time. These events therefore cause messages, which search through script code for the HyperCard objects (stacks, backgrounds, cards, buttons and fields) until a code section is found that applies to the event. If no code can be found, then a Macintosh dialogue box appears explaining that the command is not understood.

Despite its ease of use, there are limitations on HyperCard which reflect its differences from hypertext systems. For example, HyperCard has restricted database facilities, with searching confined to short character strings. Perhaps more significant is the fact that a HyperCard stack abandons the use of true windowing systems and instead uses the entire screen[19]. Although it has a superficial similarity with Object-Orientated systems, HyperCard lacks inheritance for the cards or stacks which are defined, and the stacks themselves cannot be nested.

A major problem concerns the representation of location within the stack, and HyperCard does not have any form of Graphical View Finder to help with either visual portrayal of position (a fundamental requirement for hypertext), or to allow structure editing (with dynamic updating). Finally, although the HyperTalk scripts allow a reasonable level of control of the system, it is comparatively easy to identify

18: A Hot-Key is a 'short-cut' key sequence that allows a specific task to be invoked, without having to follow a potentially extensive series of command options.

19: The SuperCard product from Silicon Valley Systems offers an extended form of HyperCard, which uses true windows on a virtual screen.

specifications which require the use of external commands, which must be implemented using ordinary programming languages such as Pascal or C.

The Guide system [Brown 86 & Brown 87], developed at the University of Kent and subsequently marketed by Office Workstations Ltd. of Edinburgh, is a tool intended to allow readers to display and also tailor on-line electronic documents according to their own reading needs. The goal was to produce a system that offered on-line document interaction that complemented paper-based documents. Providing continuous negative feedback, a simple and flexible user control mechanism, and supporting a good user interface would mean that the workstation software would help the reader work with the document.

Guide documents can incorporate glossary buttons. When activated, these cause a glossary sub-window to appear during the document reading. The sub-window is used for explaining jargon, for citations to books or people, or for footnotes.

The success of the Apple Macintosh Hypercard system provoked much research into equivalent systems for the IBM PC machines, but the results so far seem to have met with only limited success. An example of a hypertext system for the IBM machines is HyperPAD[20], which implements many of the Hypercard facilities, differing mainly in terminology rather than function. However, HyperPAD does differ from Hypercard in two important - and related - aspects. Firstly, HyperPAD avoids the problem of multiple graphics standard on IBM machines by not supporting graphics - it is text-only; and secondly, HyperPAD has ample facilities for colour usage.

However, despite having the Hypercard system as a form of 'prototype' on which to base the product, the HyperPAD package has a number of functional limitations which reduce its initial appeal, particularly with regard to its stability and reliability. As such, it is an example of an information handling system which is desired by end-users, but which suffers in its application. Accordingly, it is less likely to be applied in important tasks by the users.

20: HyperPAD is a trademark of Brightbill-Roberts.

Section 3.4: Hypermedia.

Given the recognisable advantages of functionally hypertext architectures, there is a natural derivative of the concept which extends the systems to include such additional heterogeneous components as two- and three-dimensional graphics, spreadsheets, video, sound and animation. In some ways, the hypertext concept is enhanced by the results of work on multimedia systems, while retaining the hypertext operation mechanisms. The term hypermedia is used when extending the hypertext functionality to these extended or hybrid systems. A hypermedia system should be able to create links within a database including not only text, but also complex diagrams, photographs, video disks, audio recordings and so on [Yankelovich 85]. In [Akscyn 88], the authors define hypermedia as being similar to hypertext systems, but additionally supporting:

> 'other kinds of information such as vector graphics, bitmapped images, sound and animation.'

A number of hypermedia examples exist, including MIT's Spatial Data-Management System (SDMS), and the Digital Equipment Corporation's Interactive Video Information System (IVIS), which integrates a variety of data types onto the same display screen.

Intermedia is a system under development at Brown University [Yankelovich 88], and proposes a framework for a set of tools that allow links and connections between the many possible heterogeneous data types, resulting in a larger document structure. In order for this to be implemented, there is a clear requirement for an alteration in the way that documents are perceived by the end-user and/or author. Furthermore, the heterogeneous data types require a variety of hardware devices beyond the standard system unit for computation, and keyboard with high-resolution display for input and output interaction. In practice, supplementary devices are required for the portrayal of the non-standardised data types such as sound and video.

It is important to emphasise that the Intermedia system is not intended to be an end-user application so much as a collection of tools grouped into a framework for use by end-user applications. This distinction is one that has significant implications for the work described here, and will be referred in section 8.2. For the moment, it is sufficient to note that Intermedia is in effect providing an architecture for a combined multimedia/hypertext system. In other words, by providing the infrastructure for data control and representation, it supports rather than directs the application. However, the control of the system is still based upon hypertext systems, which deliberately restrict user options in order to provide a more friendly and less confusing user interface. One can conclude that in any information processing system, the nature of the architecture must necessarily influence the potential of that system, and that restrictions on the architecture will inevitably restrict the processing capability.

The KMS system described in [Akscyn 88] is a distributed hypermedia system. The authors of this system note that:

> 'In shared hypermedia systems, multiple users may simultaneously access the hypermedia database. Shared systems may be implemented as distributed systems, in which portions of the database are distributed across multiple workstations and file server on the network. ... The database can be as large as available disk space permits.'

During the development of the system, the designers concluded that their experiences encouraged them to:

> '... practice a design philosophy of voluntary design simplicity, striving to make do with fewer concepts and mechanisms.'

Future development of hypermedia systems is discussed in the context of the 'Notecards' system developed at Xerox PARC [Halasz 88]. By examining the limitations and strengths of Notecards, the following issues were described as being important for consideration in the next generation of hypermedia systems:

1) Search and query in a hypermedia network.
Navigational access is not sufficient for hypermedia, instead effective access requires query-based access to complement navigation.

2) Composites augmenting the basic node and link model.
The two primitives of the node and the link are successful but insufficient. Halasz states that Notecards suffers from a lack of a composition mechanism, and the nearest approximation involves using a 'filebox concept' which was intended to provide a hierarchical organisational structure.

3) Virtual structures for dealing with changing information.
The hypermedia data model of Notecards is essentially static and fragmentary. Unless explicitly edited, the network does not alter. In particular, Halasz points out that a network cannot reconfigure itself in response to changes in the information it contains. A virtual structure would support dynamically determined reorganisation.

4) Computation in (over) hypermedia networks.
Hypermedia systems do not actively direct the creation or modification of the network. Accordingly, it may be necessary to augment a system by inclusion of an active computational engine for specific applications.

5) Versioning.
This would allow users to maintain and manipulate a history of changes to their network. Some systems - such as Intermedia - do provide aspects of this facility, however; only single copies of nodes are kept, but alternative versions of the entire web (the interconnection links) can be constructed.

6) Support for collaborative work.
Notecards is designed as a single-user system, while KMS and Intermedia support simultaneous multi-user access.

7) Extensibility and tailorability.
The generic nature of hypermedia systems supports a wide variety of task domains (creating, editing, displaying, storing, retrieving and interconnecting), but is not especially *good* at any specific task. In effect, the systems are useful but not well adapted to tasks. It would be preferable for users to extend systems with new functionality or to tailor existing functionality to better match the exact requirements of the application.

Section 3.5: Information Browsing, Presentation and Preparation.

Given that a suitable architecture can be found for supporting multimedia manipulation tasks, the next major problems that must be addressed concern data (or knowledge) exploration in the resulting mixed-media environment. Results of work on the problems of knowledge exploration comes from such fields as computer-aided instruction (CAI) [Osin 76] and the nonlinear text or hypertext systems referred to in section 3.3. Some database systems, such as the MIT Spatial Data Management System (SDMS), have also adopted a hypertext approach to allow users to wander among the information in a database. SDMS [Herot 80] provides a user with graphical querying and presentation of the database, and will be discussed in section 3.6.

Shasha [Shasha 85 & Shasha 86] suggests a fragment theory for retrieval and knowledge exploration in his data model called NetBook. The knowledge is represented in the forms of text fragments (including a picture, an experimental simulation, or a live performance) with relations defined among them, and queries in a natural language form helps users to access the appropriate fragments. This system clearly has a requirement for an information architecture that provides a structure for representing not only the data of either mixed or single types, using concepts that resemble in some ways the NetBook text fragments, but also the relations between the fragments. In practice, however, the primarily text-orientated model offers rather limited control or variety in presentation content, and minimal interaction options.

A complex mass of information can often be more clearly presented with the appropriate use of colour and dynamic representations which are available on most examples of graphics and engineering workstation technology. The graphics editing system developed by Feiner, Nagy, and Van Dam at Brown University [Feiner 82] uses this approach in preparing and presenting technical manuals. Their system uses colour, very high resolution graphics, and menu selection to provide flexibility in scanning and re-presentation of a document. Brown and Sedgewick of Brown University have demonstrated the ability to see dataflow and control structures of algorithms and software as they execute in their algorithm simulator and animator, BALSA [Brown 84]. BALSA uses a dynamic graphic interface as the natural mode of interaction. All these tools aid software engineers and computer scientists in understanding and 'feeling' the action of software or algorithms.

Another important and relevant research issue concerns systems used for the preparation of documents. Examples of existing systems that allow for interactive document editing and formatting operations are Janus [Chamberlin 82] and Andra [Gutknecht 84], of which Andra provides additional hardcopy capability on a laser printer. Other document systems like CD-GUIDE [Brown 86b][21] allow for interactive viewing and storage using a CD-ROM disk.

A renewed interest in text processing research has resulted in new text editors and document formatters producing integrated programs that format documents dynamically during the editing session. Examples are described in [Furuta 82a], [Gutknecht 85], and [Peels 85]. General hierarchy editors for documents, programs, and graphics are in existence.

21: An extension to the GUIDE Hypertext system described earlier.

Hierarchy editors consist of interactive programs which browse through and modify, tree-like structures of data. Text-based editors such as Walker's 'Document Editor' [Walker 81] and PEN [Allen 81] are used specifically for document editing. Some editors (such as Walker's) include facilities for cross-referencing to show inter-structural linking, though these are not explicitly shown in final presentations of the hierarchical structure. Both cross-linking and topological data organisations can be derived as natural products of one uniform structure, as seen in other systems such as ZOG [McCracken 84] and Textnet [Trigg 86].

The design of a user interface management tool to improve human-machine interactions is also an important concept. A discussion of User Interface Management Systems (UIMS) is presented by Olsen et. al. in [Olsen 84], and identifies some of the major problems that UIMS research should be addressing. Other discussions on the UIMS concept can be found in the papers [Thomas 83], [Enderle 84a] and [Enderle 84b]. Interesting topics covered include: the interface of the UIMS to the application and graphics; the structure of a UIMS; and special CAD aspects of UIMS.

Traditional document systems show very little of their internal state to the user. Developments in screen editors, spreadsheets, and electronic desktops are making use of powerful graphics on workstations to show as much of the system state as possible on the computer screen. Examples are Xerox Star [Smith 82], Apple's Lisa and Macintosh machines [Ehardt 83], Alis [Applix 84], and Framework [Harrison 84]. A concise review of the current approaches to the use of windows in screen dialogues by application systems can be found in [Konsynski 85].

Commercial document environments like Framework by Ashton-Tate and Symphony by Lotus Development Corporation [Jadrnicek 84], and others, provide extended capabilities, such as spreadsheets, word processing, database management systems, graphics, and communication software. Framework provides an 'outline' function that allows users to create documents with an outline; entries in the outline, such as a graph or a spreadsheet item, can be imported from different component packages. The way these systems integrate different tasks has caused an imbalance in that each package has grown around the success of one subsection of the task activities (for example, spreadsheet or word processing), and the others are only half-heartedly supported.

We have seen that an important aspect of information presentation concerns the tools used to support the presentation. Extending this further, we must mention the current research into Graphical User Interfaces (GUI's). A loose definition of a GUI is that it is a user interface that operates in a computer's graphics mode. However, this can exclude some packages that would otherwise be categorised as GUI's[22]. The Microsoft company has become very interested in GUI concepts, not least because it is a primary developer for the OS/2 Presentation Manager package, which embodies many of the fundamental GUI concepts. Microsoft has defined a true GUI system as satisfying six requirements [PCMag 89]:

1) It exploits bit-mapped displays, offering true WYSIWYG screen representation of printed output.

2) It is a graphically orientated interface that makes extensive use of icons.

3) It has good screen aesthetics: it looks good and is a pleasure to work with.

4) It allows direct manipulation of on-screen elements.

5) It embraces the object-action paradigm[23].

6) It offers standard, expected elements such as menus, windows and dialogue controls; in order to provide consistency across applications.

22: For example, Desqview from Quarterdeck, or the HyperPad product discussed earlier.

23: The user chooses an object first, then selects the action.

The above list is rather subjective, considering its source, and is rather selective in describing what a GUI is. By including such elements as multiple-application support, and inter-application communication, we can envisage more accurately the facilities expected from a GUI. In theory, a GUI should make applications more powerful, not least because of the common application interface. However, many applications are sufficiently complicated that it would be impractical to standardise on a specific interface orientation.

Despite this, the goals of GUI's - to save time, to make programs easier to use and more powerful, and to ease the learning curve - are all important features that are demanded more frequently from current applications.

Research into GUI's is being carried out and has resulted in a number of applications, including NewWave (Hewlett-Packard), Open Look (Sun Microsystems) and OSF/Motif (Open Software Foundation). NewWave in particular is interesting, because it runs under another GUI, the simpler Microsoft Windows package. It incorporates an object-orientated paradigm into its construction to extend the normal level of GUI capability. As a simple example, by moving a document icon to a printer icon, the document will be printed without further operator control being required.

Section 3.6: Information Structuring.

The drawback of existing integrated editor/formatters is that the high-level structure of the document is not well-represented [Furuta 82b]. Aldus Pagemaker on the Apple Macintosh is a good example, in that it exhibits cut-and-paste operations in preparing mixed-media documents, but with no proper support for overall data structuring.

Hewitt [Hewitt 77] demonstrates the use of 'actor message passing' in viewing control structures, where the actors are purely active objects with no recognition of any data structures. The OPAL system [Ahlsen 84] has the same focus with the concept of a 'packet' as the principal data and action structuring mechanism. A related area of research into actors formalism is used in a programming system by Byrd et. al. at IBM Thomas J. Watson Research Center [Byrd 82].

Taking the concept of information structuring to a more formal level moves us into the domain of databases, and particularly relational database systems. Notably, the earlier work of E. F. Codd [Codd 70] in developing a relational theory for data was to be used as a framework for database systems. His model covered three aspects that all database management systems have to address, including integrity, manipulation, and importantly, structure.

Structuring of information is particularly important if the data is being used within a distributed environment. The structure must necessarily reflect limitations in the storage mechanism unless the structure is defined in terms of virtual storage - a concept that is as yet undeveloped. Clearly, centralising the information storage makes backup and transaction processing more straightforward. Centralising also has the immediate drawbacks of reducing localised control (implying reduced independence) and further a dependency that the central information store remains operational at all times.

Sarin et al have considered the processing of data updates in a database architecture, particularly in a distributed architecture [Sarin 87]. The problem is to ensure that data updates which arrive at a site must be saved in a fashion that preserves mutual consistency, while at the same time allowing for removal of sites that are excluded for reasons of site crashes or other causes. In effect, the multiple sites must be able to allow updates at any time, and to reconcile database copies after they are notified of updates at other sites.

There are examples of information processing systems that have been developed with minimal concern for the actual storage of the potentially large scale quantities of varying data types. Mayer [Mayer 85] describes work carried out during the development of a computer conferencing system, in which emphasis was initially upon the user viewpoint:

'... we did not immediately start worrying about file layouts, record structures and the like. We started thinking about how the user would view and use the system and then worked backward from there. Of course, the design process is never quite that simple. We had to go through several iterations, because something that we'd thought might be a great idea at the user level turned out to be impossibly complicated to implement but could be done easily with a small change to the user interface.'

This strategy of developing first the specification of the system and then using this to design the architecture - with inevitable requirements for re-design - is one that was applied for the prototype system described in this thesis. Mayer recognises that this ideal is more difficult to achieve in practice, but an analysis of the various information processing strategies is very helpful in deciding upon the primary requirements for a general information processing architecture. The importance of the storage architecture is further acknowledged by Mayer:

'In a system of this sort, the files created by the prototype tend to get inherited by successive generations of the software. Think how annoyed users would become if the old conferences became unreadable whenever a change to the software was made. Thus, the organisation of the information (storage architecture) in the system can have profound implications to later development of the software. You can't just say (that) with version 4 we'll go to a totally new file organisation because then conversion programs will need writing, and some of the imaginable conversions may not even make sense because certain necessary data was never stored in the original files.'

As mentioned in section 3.5, Herot has discussed the concept of spatial data management, describing it as a technique for organising and retrieving information by positioning it in a Graphical Data Space (GDS) [Herot 80]. In contrast to conventional database management systems, a spatial data management system (SDMS) presents the information graphically in a form that encourages browsing and requires less prior knowledge of the contents and organisation of the database. Herot summarises the differences between spatial data management and standard database management systems as follows:

1) Motion through the database is simple and natural. A single control, the joystick, allows the user to explore the entire database. Conversely, conventional symbolic query languages require the use of special syntax and semantics.

2) The database is its own dictionary. Conventional DBMS require a data dictionary to define the structure of the database. In the SDMS, the graphical data space is its own description, and rather than specifying a relation and attribute name, the user traverses the data space until he reaches the desired information.

3) The presentation of the data encourages browsing. An SDMS display almost always gives more information that is needed. In conventional DBMS, every piece of data must be explicitly requested, and it is difficult to place related data together. An SDMS allows the database administrator to arrange the information according to any chosen attributes.

4) The placement of data can convey information. This works in much the same way that a person can find some needed information by recalling where the paper is that has the required information. Examples of this concept in an SDMS include the arrangement of data according to freshness or priority.

5) Graphics can be used to convey information. The most familiar forms are histograms and graphs. Representations can also be used to depict trends that would otherwise be hard to formulate into symbolic queries.

6) The system can accommodate new data types, such as photographs.

Section 3.7: Computer Architectures.

In several of the current systems, there has been the implicit requirement for further devices beyond those available on most workstations. As well as the now standard workstation combination of keyboard, mouse and high-resolution display, there is a need for such peripheral devices as sound, video, and graphics drivers. Currently, computer architecture - particularly that of workstations - makes no provision for such features as part of the orthodox system paradigm. Accordingly, where such mechanisms are to be attached, the resulting connection is not optimised for the architecture, with resulting delays to the operation of the devices.

Such difficulties are especially prevalent when the system is intended for multi-user application, and large quantities of data must be transmitted at very high speeds. The extremely high bandwidth required for - say - video signal exchange precludes all but the highest performance data transmission systems from accomplishing the desired results.

Much research is directed towards the problems posed by Distributed Multimedia Information Systems (DMIS) [Aceves 85b], particularly for those environments where workstations are to be connected using high-speed local area networks. Such systems have two main requirements:

1) Provision for store-and-forward data interchange (such as electronic mail), where the interchange is not required on a real-time basis.

2) Real-time interchange (such as for conferencing), where immediate response is required.

A good summary of the problems and issues can be found in [Ngoh 89], which concludes amongst other points that standard data transport technologies:

> 'do not provide an adequate mechanism (for information interchange between DMIS workstations)'.

Nor is a broadcast technology sufficient. Although such a system would have the advantage that only one transmission is required to send data to those hosts participating in the DMIS, it follows equally that hosts not participating in the DMIS, or not requiring the transmitted data, have a considerable increase in their workload in order to identify and filter out the extraneous material. Unfortunately, DMIS tend to produce large quantities of data that would require broadcasting, so the issue is not one than can be discarded as being of minimal implication.

With regard to the actual processing of data, there is little doubt that a multi-tasking architecture is mandatory as the operational paradigm. The more important question is regarding the nature of the multi-tasking implementation. Current popular realisations of the concept are based upon the simplistic model of a single central processing resource allocating units of job time to a series of tasks. As the number of tasks that must function concurrently rises, so the inadequacies of this approach become manifest. Furthermore, for a single processing unit to be capable of executing all the possible jobs that will be required, the processor must be generalised in its capability - regardless of whether it is based upon CISC[24] or RISC[25] technology.

However, if it is accepted that significant numbers of multi-tasking jobs require only a limited instruction processing capability, and further that the individual tasks executing under the multi-tasking environment are modest and concise, then the need for complex processors is reduced. Instead, it is perhaps worth considering the alternative of a multi-processor computer architecture, with a large pool of elementary processors for the normal and simplistic tasks that form the 'bread-and-butter' of

24: Complex Instruction Set Computer.

25: Reduced Instruction Set Computer.

multi-tasking computation, and a smaller administrator pool of more powerful processors to coordinate and control the lower level processors.

The problem then arises of how to control the resulting architectural complexity. As [Randell 86] indicates:

> 'The task of implementing a large and sophisticated computing system is often unduly costly and time-consuming, with the resulting system exhibiting inadequate performance and reliability, because of excessive system complexity.'

Recognising that a good way to cope with system complexity is through the principal technique of 'divide and conquer', Randell states that:

> 'Complexity can be reduced significantly by ensuring that the system is constructed out of a well-chosen set of largely independent components, which interact in well-understood ways. However, the task of structuring a system, i.e. of choosing and defining appropriate components, can be very difficult.'

The problem is considerable, and one of on-going research; not just in terms of pure computer architecture, but also with regard to network-based distributed systems. Randell proposes a system based on a hierarchical approach, with specific research work implemented on an addition to the UNIX file system structure. The file-naming scheme within the system is extended up beyond the hosting machine so that the file systems are further categorised by machine, department, and so on. Naturally this raises further problems of security, performance and reliability; but these issues fall outside the scope of the current discussion. Randell's work emphasises the following points:

1) The principal system component is an 'ability' that: "I have termed 'distributedness', which makes a single set of hitherto independent systems function as a single coherent system."

2) That the approach taken to provide system abilities is one of 'simplifying generalisation'

3) That the computer science world has concentrated unduly on the design of sequential and centralised systems: "This concentration is perhaps the true Von Neumann bottleneck."

4) The environment of people, procedures, machinery, etc., that a typical large-scale computing system is intended to fit into, and contribute to, is itself a distributed system, usually possessing an enviable degree of flexibility and resilience.

Section 3.8: Groupware.

The purpose of Groupware is to provide both structure and support mechanisms to aid users in working together. Accordingly, [Tazelaar 88] defines it as either 'software for a group' or alternatively 'computer-supported cooperative work.'

Engelbart et al discuss the groupware concept under the name of Computer Supported Cooperative Work (CSCW), and define it as dealing with the study and development of systems that encourage organisational collaboration. They describe CSCW systems as falling into three categories [Engelbart 88]:

1) Tools for augmenting collaboration and problem solving within a group geographically co-located in real-time.

2) Real-time tools for collaboration among people who are geographically distributed.

3) Tools for asynchronous collaboration among teams distributed geographically.

Engelbart's system (NLS/Augment) was designed to support members working in varied disciplines, including software engineers, managers and social scientists. The underlying architecture was hierarchically structured, with the file structure separated from its content. Originally, nodes were strictly textual in nature, but later referred to a 'property list' of content nodes of varying types, including other hierarchies. The goal of developing the system was based on the belief that creating tools for collaborative knowledge work was essential to the necessary evolution of work groups in increasingly knowledge-rich societies and for increasing organisational effectiveness. Until the recent interest in CSCW or groupware systems:

> '... most developers limited their analyses to technical issues and ignored the social and organisational implications of the introduction of their tools; such considerations were, however, key to our work.' [Engelbart 88]

Engelbart et al go on to discuss the value of such systems:

> 'There is growing recognition that some of the barriers to acceptance of fully integrated systems for augmenting groups of knowledge workers may be more significantly social, not solely technical. The availability of rapidly evolving new technologies implies the need for concomitant evolution in the ways in which work is done in local and geographically distributed groups.'

Going further with regard to the issues of integration, the writers state that:

> 'Aspects other than introducing new technological tools into the work space (e.g. conventions, methods and roles) are at least as important to the success of any CSCW system. The elegant tools available now and in the future - superlative graphics, artificial intelligence services, and so on - only make sense in an integrated workshop of tools in which information may be exchanged. The tools in such an integrated workshop need to be conceptually and procedurally consistent.'

In effect, Engelbart et al are agreeing with many other researchers in regarding a generalising of information processing and handling strategies.

Work by Terry Winograd puts the groupware concept into the perspective of communication issues:

> 'Whenever you convey information, you are embedded in a context that makes it relevant to getting something done. The meaning of the words serves as a starting point for interpretation that leads to networks of interconnected actions. By directly addressing this universal dimension of human communication, we are beginning to develop groupware that offers a new simplicity of design and effectiveness of management.' [Winograd 88]

Section 3.9: Summary.

Much of the work of the researchers described here has focused on diverse issues. However, from a more general perspective, there are a number of conclusions that have been reached by each research project that can be applied across a much wider range of applications, thus having significant effect on the development of an enhanced information system architecture. Such a structure will require consistent mechanisms for control and data organisation, as well as the integration of components other than the basic technological tools.

During the discussion of multimedia systems, a series of five points were raised which represented definitive characteristics for such systems. We can reiterate the points as follows:

1) Promotion of connections between ideas and comments.

2) Support for flexible control and presentation.

3) Flexible editing tools.

4) Flexible retrieval, reading, and searching capability.

5) Preservation of historical integrity.

These issues are clearly of relevance, not just to multimedia systems, but to any system purporting to provide information processing support. Some aspects may be less important than others in certain circumstances, and vary depending on the focus of the application system itself.

For example, hypertext systems are particularly concerned with the way in which the linkage mechanism between the data fragments can be used in a variety of ways according to the users preferences. The nature of the links is rather simplistic, but does indicate the value of the property of a relationship existing between items of data. Furthermore, links hint at the fact that relationships are of different value to different users - some links are followed, others are not, according to requirements. Correspondingly, relationships may mean something to certain users, but nothing to others.

As [Brown 87] suggests:

> 'If presentation software simply tries to imitate a paper document on the workstation screen, we believe that, in spite of future hardware improvements, readers will continue to prefer paper. The reason is simple: the imitation will always be inferior to the real thing.'

By way of contrast, a system for on-line documents can offer great advantages as a result of the facilities for interaction. Multiple windows allow the comparison of different pieces of information.

Herot's work gives a large number of clues to the structuring of data within an information architecture, in such a manner as to offer a number of advantages to the ways in which the data can be organised, accessed and extended. Taken in conjunction with Randell's preference for simplifying generalisation, we are in a position to work towards the derivation of an enhanced information architecture.

Chapter 4.

Consideration of Current Systems and Related Research.

Section 4.1: Overview.

In this chapter, we are interested in considering the implications of the results of current work on computer-based information systems. The introduction of potentially powerful generation systems has seen only limited success, despite the fact that they should in theory produce more capable systems at reduced cost or effort. Even systems such as C.A.S.E. tools, which incorporate aspects of application purpose as well as application function, have been less successful than had been hoped.

In part, this may be seen as being a result of pre-occupation with compatibility while steadfastly refusing to backtrack and consider implications and applications of lessons learnt. Integration is often promoted as being 'a good thing', but in a manner which implies that a fast car is 'better' than a farming tractor because it is more 'flash', although the fast car is probably incapable of ploughing fields. In other words, integration is often viewed as a good idea, because it makes data exchange easier. But little consideration is given as to *why* the data exchange is being performed, or the implications of the exchange. The support for integration is generally at a very low-level within a system, which means that effects on the higher-level abstract tasks are poorly planned.

Integration is sometimes viewed as being a prototype mechanism for supporting computerised lateral thinking. By making provision for cross-connections between multiple applications and multiple users, it is hoped that good ideas may result. But using existing analogies as the first design stages is not necessarily a good thing. An electronic book would be severely curtailed if it could only mimic a paper-based book, because of the exclusion of potential cross-referencing advantages.

Similarly, attempting to use the human brain as a processing model implies that we are trying to enable computer systems to imitate human thought processes, which may not necessarily be a productive approach. Management concepts generally recognise the potential advantages of assigning groups to certain problems, rather than individuals. A group or team may be able to draw on individual talents, expertise or experience in assigning specific sub-tasks or problems, so that the overall efficiency of the team is improved. However, this relies upon the fundamental assumption that the team can be organised in a manner conducive towards cooperation and productivity rather than constraint and inaction.

This means that for problems where multiple tasks can be identified, suitable for solution by multiple processes, there must be a coordinating mechanism to ensure successful integration of the activities. The coordinating mechanism itself will be dependent upon the fundamental understanding of the problems of group organisation. Thus, simply forming groups of people to solve complex problems does not of itself guarantee greater productivity or a successful result. Similarly, constructing computerised systems that attempt to mimic human processing by duplicating specific processing tasks cannot guarantee any benefits, because there is as yet no real understanding of the coordination or integration tasks that we perform automatically.

However, the concept of integration does have direct relevance regarding the arrangement or structuring of data. In order to data to flow, there must be reasoned organisation to the routing or paths. The implications of this in terms of results and conceptual re-adjustments must be incorporated: this requires that feedback be modelled and incorporated. Understanding these issues would appear to be vital if we are to attempt the construction of systems that *support* our work.

Section 4.2: Introduction.

We have now considered some examples from the range of current products and research projects, each of which addresses or uses specific instances of the many tasks of information organisation and processing. Some of the products described attempt to provide a generalised package that deals with problems as a whole - other systems are intended to contribute to the overall solution, but leave the specification and implementation of that solution to the user or to other packages. Similarly, the research material discussed tends to address specific issues or cases within larger scale topics. The projects have normally been very successful in achieving the target goals. However, the act of establishing a target goal means that an end-point has been identified, and upon reaching the goal, the work will often be deemed complete and so is concluded.

In this thesis, we will attempt to go further than the original end-points by considering in some detail the material conclusions of the various research studies. Significantly, we will discuss the work from a perspective that is not limited by focusing on the original study direction. Using this approach, it should be easier to recognise and evaluate some of the implications of both intended and accidental study results. This will help with the task of identifying alternative applications and contributions from the research conclusions.

In this chapter, we will be looking in more detail at those characteristics of current systems and research which relate to general information administration and handling. This will enable us to work towards identifying the problems that are the focus of this research, and subsequently lead to the development of possible solutions. Consideration of existing products will help to identify not the actual attributes of what an information system should offer, but instead to evaluate what attributes are actually desired in the light of the success (or failure) of specific systems.

Section 4.3: Discussion of Current Systems.

A common and key feature of the example systems is that they are all 'self-interested'. In other words, it is assumed that the system is intended as a clearly-defined component in a task, and may even attempt to describe the entire solution for carrying out that task. Such products can be labelled 'focused application systems' as distinct from the more open-ended 'diverse application systems'. Fourth

Generation Languages (4GLs) and Applications Generators are good examples of focused systems, although at first sight they might appear to be more 'cosmopolitan' in their approach. The purpose of the packages is to attempt to specify as much as possible of the final program code for an end-user application. Yet in most cases, the 4GL or applications generator never produces more than 80% of the final code, and even that is produced by working with established modules [Gibson 89]. The remaining 20% of application code still has to be carefully appraised by a human system developer - at best to fine-tune its performance, and at worst to re-engineer from scratch using guidelines issued by the initial generator. Thus such packages could be viewed as little more than advanced library managers that have analogies in the object-code handlers of straightforward program linkers. Furthermore, the material produced by a 4GL or Applications Generator is frequently self-referencing, imposing limitations on the customisation that can be carried out by developers.

This inflexible approach for system development can also make it very difficult for the end user - who should in fact be the main focus of interest and support - to develop a complete system with the required flexibility for the task at hand. Only in certain special cases will development systems acknowledge the existence of other packages, a characteristic which is even more prominent for packages from different software houses.

The exceptions tend to involve recognising the existence of indisputable standards, such as the Lotus or dBase programs on the IBM compatible machines, and an acceptance of the standard format used by the data files of those programs. However, the acknowledgment of such a standard is usually rather 'grudging', and typically takes the form of a data conversion process that is typically one-way. For example, Ventura Publisher, a powerful Desk-Top Publishing package for the IBM machines, is perfectly capable of incorporating text or graphic images from 'external' packages into the specific document; but once those components have been correctly integrated within Ventura Publisher, it is not particularly easy to 'export' the components out again as they will have been 'glued' together into a larger (and hopefully better) whole. Other packages will often provide conversion programs that convert external data to the preferred format. For example, the IBM-based Word

Perfect word processing package provides a multi-purpose conversion program to convert data files created by other word processors. But the conversion program only works one way.

An alternative approach is to try and standardise on an agreed format for data, and this has been the objective of the ODA and SGML document description standards. Any systems that work using these standards should be able to successfully exchange documents. However, the origins of these two standards places restrictions on their suitability for general purpose data exchange, particularly with regard to those processing tasks which do not necessarily have a meaningful representation as a document - for example compiling a program.

Moving to consider the issue of development systems such as C.A.S.E., there is little doubt that the objectives of the concept are earnestly sought after, particularly in view of the generally recognised backlog in development of software [Akass 89]. The problem is that the current implementations of C.A.S.E. concepts suffer from over-promotion coupled with a lack of tool standards and integration. According to Cally Ware [Ware 89]:

> 'Too much has been written about how C.A.S.E. will solve all your software development problems at a stroke. How it is, in short, a panacea that will automatically compensate for poorly designed systems, untrained or unmotivated staff, or a lack of real commitment to get things right, now.'

In order to be of benefit, Ware suggests that it is important to recognise that:

> 'C.A.S.E. is not simply a collection of tools. It requires a combination of skills and good management to make it work.'

Amongst other vital factors, Ware includes the following as being essential to the success of C.A.S.E.:

1) A methodical, structured approach to development is essential; and must be rigorously enforced.

2) Any software project must be managed.

3) A C.A.S.E. tool that will not easily link to other tools (word processing, desktop publishing, project management, application generators, mainframe dictionaries, etc.) will only ever be of restricted usefulness.

The concept of C.A.S.E. is fundamental in depicting an important aspect of information systems. No matter how good the underlying tools used for processing the raw data, this can all be wasted if the higher level administration, control and integration tasks are neglected. C.A.S.E. does not of itself have a direct relevance to information systems, except that it embodies a high-level administrative function of the kind that must be present in any information system of value.

Inevitably, however, as a focused application system becomes more detailed, so the data and control structures become increasingly low-level but complex, with a corresponding reduction in export capability. If there exists a clear industry standard (such as Lotus 1-2-3 for spreadsheets on personal computers) then there is concerted effort on the part of system developers to permit relatively easy accommodation of the standard as a sub-set of the product capabilities. But for applications which do not have a notable market leader (for example, the 'Personal Information Manager' applications on personal computers, or C.A.S.E. tools on larger systems) then there is little in the way of consensus in development of standards or even agreement on exactly what facilities should actually comprise the application. Each developer is attempting to produce the definitive program or package that will become the standard in that application.

Nevertheless, the desire for greater cooperation between packages running on more powerful computers has led to the development of extensive and advanced architectures such as the PC multi-tasking system Microsoft Windows and its derivative

OS/2 [White 87], as well as the associated Dynamic Data Exchange (DDE) concept [Schifreen 88]. Any application package which is intended to run successfully 'within' Windows must be capable of handling DDE satisfactorily. This enforces a degree of integration between packages. DDE has a number of defined data exchange formats, any of which may be used between any two existing Windows applications.

However, the DDE concept only provides for a higher conceptual level of raw data exchange. This is insufficient to provide for the exchange of information, because data alone - no matter how detailed or complex - is not a complete definition of true information. Accordingly, while DDE is to be welcomed for its advance upon existing systems, it must be acknowledged that there is much work yet to be done to extend integration beyond mere data exchange.

Thus, current systems based on clearly defined applications (such as spreadsheets) still bypass the major issue of the integration of information handling processes. This means that although the raw data can now be exchanged and processed reasonably well, there is little or no cooperation between the applications themselves with regard to the data. Typically, at any one moment in time, only a single application will have 'exclusive' rights to a set of data - it will define the format used for storage of the data, and only that application can be 'trusted' to modify the values in the data set.

The problems of sharing form the subject of considerable research, and this is particularly true with regard to the sharing of 'resources' available on computers - especially multi-user systems. One approach used is to view the mechanics of resource access as being similar to data access mechanics. In effect, the representation of resources can be handled as a form of data type (in much the same way that certain operating systems[26] treat all devices as files). Considerable research has gone into the complex intricacies of resource sharing, but generally with emphasis upon the particular problem of resource control which is required to prevent the 'deadlock' situation, where two or more applications both seek access to a given resource, but will not yield

26: For example, UNIX.

the rights they currently have on other resources. Thus this research is only of limited assistance to the problem of data exchange. For multi-user and multi-processing systems, where our definition of resource can be taken to much higher and more abstract levels, there is a need for study into the problems of rights of access, ensuring integrity of the resources, and consideration of what forms of access are considered valid, as well as when they may be permitted.

Section 4.4: Discussion of Current Research.

More successful aspects of modern application systems can be found in those technologies that have evolved out of meticulous research[27], and which have a more considered viewpoint for the end-user. Many of the systems go to great lengths to ensure that the user is given a more coherent presentation of the data - reflecting the importance of the environment in which the user is working. However, systems based on - for example - hypertext or multimedia data architectures do not always take to the ultimate conclusions all the analysis and development ideas which are necessary to provide the user with a truly enhanced environment in which to work.

In spite of some of the current trends in research emphasis, it is not enough merely to try and 'computerise' existing techniques for data handling, because existing 'normal' computer data types[28] are low-level and limited in their scope. This remains true even when the manipulation of data types is performed within the context of 'integrated' systems. There are strengths and weaknesses to both static print and dynamic electronic media handling, not least in regard to the tools needed for performing the tasks necessary for production in each area. Much work has been done with regard to computerising the data normally held in paper form; the primary objective being the development of the 'electronic book'. A major trend of research in this area [Yankelovich 85] is to identify the strengths and weaknesses of existing paper-based books, and use them:

27: For example, Hypertext systems (Hypercard), Parallel processing (the INMOS Transputer), and Object-Orientated systems.

28: Numbers, Characters (or strings), and Logical (boolean) values.

'to formulate the capabilities that electronic document systems should have in order to maximise the advantages of the electronic medium and overcome some of the disadvantages inherent in the print medium.'

In other words, the attributes of paper-based documents (both positive and negative) are being used as the first specification of the capabilities of the electronic book. This is sometimes done without questioning the validity of such an approach, or regarding the possibility that subtle but implicit restrictions could be incorporated into the elementary design. There is little consideration of the fact that such a strategy offers little opportunity to introduce the potential improvements facilitated by computer systems, and which must necessarily be incorporated at an early stage of the system design. Yankelovich et al point out that the most important property of paper-based material - such as books - is that the data is static. Once committed to paper, a book cannot be altered except by reprinting or physically damaging the record; normally, readers are not able to change or manipulate the contents. In their study, Yankelovich et al conclude that:

'the static nature of books is both their biggest asset and their most serious shortcoming.'

For some reason, this statement is not fully justified, and the only apparent examples of the disadvantageous nature of static print offered are that it cannot handle sound or motion, and that it is difficult to create multiple indices. There is minimal comment on the advantages of static existence, beyond the mention that integrity of information results in (potential?) historical value, and that providing information integrity within a system can reduce problems of unreliable hardware. Other advantages would surely range from the trivial (relatively impervious from damage by coffee spills) to the significant (legal admissibility), not to mention the simple aesthetic benefits and convenience of being able to hold a printed page, the ease of scribbling notes, and the extremely low cost.

Disadvantages of paper documents would primarily emphasise issues relating to the need for skimming or browsing. Readers often require merely an overview of a text, or quick and easy access to relevant material in depth but which is not obscured by irrelevant detail from other topics. Indexing, good structuring and complete tables of contents help reduce the dilemma, and an abstract is good for an overview - but each scheme depends upon someone else's viewpoint, and so remains as merely a patch for the problems.

However, the advantages of an electronic book were much easier to define, having been described as early as 1968 [Engelbart 68]. The benefits of using computers in the construction of electronic books can be seen when they are used for creating connections, or 'webs' between related material. With normal printed text, ideas which result from the reading should ideally be recorded somewhere, and typically they are kept as annotations next to the material that induces them, or as totally separate records which may subsequently be lost.

Yet both of these have the disadvantage of not promoting easy cross-referencing, nor is it a simple matter to locate such notes at a later date. Consider as a single example the loss to mathematicians of Fermat's last theorem[29]. Many researchers have stressed the benefits of working with connections between data, and the value of ideas leading to the derivation of new connections. The clear implication is that material held by an electronic book would be much more suitable for linking with connections, because of the potential for flexible organisation of material. It is this concept that subsequently introduces the idea of hypertext.

Yet the intrusion of enforced structure into the material can also result in less flexibility for the computer based document, when compared with the paper document. According to Hansen et al [Hansen 88]:

29: Fermat's last theorem was a proof of the hypothesis that the diophantine equation:

$$x^n + y^n = z^n$$

only has integer solutions for n = 2. He noted in the margin of a text that he had found 'a marvellous solution to the problem, but do not have space here to record it'. For many years, mathematicians expended considerable effort in trying to reproduce the proof, but without success. Current thinking is that Fermat's proof may have been flawed. The loss to mathematics could therefore be described in terms of effort rather than result. Equally, however, the effort resulted in the gaining of many other theorems and a greater understanding of the subject.

'Several factors can influence the behaviour of users as they read and write with computers.'

Such factors include page size, legibility and responsiveness. Yet it would seem that the links and structure of the computer-based document, and the mechanisms used to follow them, are more significant in channelling the options available to the reader into a specific but limited set of possibilities at any one time.

Once the webs of connections have been established, it is possible to use the computers to focus on the connections themselves rather than the data that they interconnect. This enables an 'overview' of the data to be considered, providing a visualisation of the structure of the data. Beyond the manipulative advantages of computer-based electronic books themselves, comes the additional and significant potential for supporting inter-communication between users of the material, whether they are originators (the authors) or everyday readers. Clearly, this offers much in the way of enhancement for learning, discussion and general research; but there is also the issue of information integrity, and this becomes a more significant problem with the incorporation of the 'multiple book' concept. An electronic book could come in a number of 'editions', each of which could be selected or viewed by multiple readers according to need or preference. Extending the idea, academic (textbook) material would rarely be considered complete, but instead viewed as offering only a limited contribution to a potentially massive linkage. In order for this extensive data group to retain its consistency, revision access must be directed or controlled. Therefore, it should be possible for the data to have associated attribute records, which - amongst other things - will describe the various permissions that the reader or reader category has with regard to that data object. Such attributes would enable a 'filtering process' to identify material that can be altered by the reader.

Such a system naturally lends itself well to placing restrictions upon certain sections of the data (perhaps for security reasons), since unauthorised readers would not be permitted to retrieve the secured sections of data. This further means that multiple readers with selective read- and write- permissions can have varying views of the electronic book - in effect they would have different 'editions' of the same book.

However, this same flexibility introduces potential problems with regard to the organisation of the data, as recognised by Yankelovich et al:

> 'as the number of connections and quantity of information increases, so does the difficulty of generating maps of the entire information web.'

Indeed, it is perfectly possible to envisage circumstances where the resultant data 'webs' could be considerably more complex than the data that they refer to, and in an expanding state following multiple user accesses and manipulations or updates. This is a major problem that overshadows all current work, and as yet has no straightforward solution.

In a sense, multimedia systems are the precursor to hypertext systems, even though hypertext as an idea has been around in one form or another since around 1945 [Bush 45]. The main difference is that multimedia systems are essentially directed towards the simply specified task of handling potentially large quantities of varying (multimedia) data types. The actual usage of the system, and indeed, the presentation of the manipulated data, is either left almost entirely to the user of the system, or limited to falling within exact constraints imposed by the system developers.

In a similar way, despite the great lengths to which hypertext developers go to ensure that the presentation of data is easy, natural, and intuitive; there is still the underlying concept of treating data (normally text) in lumps for presentation 'a page at a time'. The justification for this is that if information takes more than a 'page' to present, it probably needs to be broken down further still, into smaller pieces. This is deemed necessary in order to make it easier to find a way around the data, thus establishing paths that follow links between the sections of data. As evidence of this point, it is generally recognised that much hypertext research is addressed towards the presentation of only a partial component of the hyperverse[30], instead of first considering the relationship between data presentation and the end-user 'assimilating' the data.

30: The HyperVerse is the logical extension of Hypertext systems, whereby all Hypertext systems will merge and handle all data and media types.

Additional questions of complexity can rapidly arise within a hypertext system because of the ease with which jumps across the links can be made. Engelbart [Engelbart 87] has suggested that in fact, a 'hypergrammar' should be developed in order to constrain the authors from 'leaping around'. It may be true that human perception of 'information' is 'spaghetti-like' in form, yet to minimise user confusion, the presentation of that information should be as straightforward as possible - multiple-link possibilities will quickly begin to confuse the presentation by literally drawing a 'web' over the data. These two aspects introduce a paradoxical difficulty: to represent a complex network of data without using a complex net for exposition.

Furthermore, tools for the formation and use of hypertext systems are still at a very primitive stage. The ability to search structures, to check links, and to detect 'stubs' within links is of great importance, but as yet there is no hint of an emergent technology for the development of such tools. Despite the widespread use of word-processing technologies, the strategies for the construction and development of documents are derived primarily from publishing technologies dating back centuries. And yet there is already a serious (and impatient) proposal to jump ahead to work on the hyperverse concept, which will extend the ideas of hypertext to all media types. Without a better understanding of the principles embodied within hypertext system creation and access, there is a danger that hyperverse systems may be constructed upon inconsistent or inadequate foundations. A question that must also be answered as a matter of urgency concerns the practicality of working with the large systems implied by a hyperverse architecture. The larger the system, the more likely that there will be multiple contributing authors importing multiple data types, and this introduces significant problems concerning the integrity and coherence of the data which is to be integrated into the resultant document system.

As indicated in section 3.6, many systems are developed using specifications that arise from examining the characteristics of existing concepts. Although this has the advantage that much of the basic work stems from familiar circumstances, in that the objectives are initially definable and tangible; there is nevertheless a contradictory component deriving from the tendency to implement systems in terms of generally well understood implementation techniques and characteristics. In other words, if an existing system has any basic components in common with the desired application, then it is natural for these elementary units to be incorporated. However, the ease with which this can be done means that often, these fundamental components are ducted artificially into structures or circumstances for which they were never intended, and in order to achieve the modifications, the unit could conceivably be compromised with regard to its validity and/or accuracy. An example of this could be found in some spreadsheet systems, which are 'enhanced' to include database facilities. The database functions operate by treating the spreadsheet cells as record fields. However, many spreadsheet functions are capable of modifying entire groups of cells, and there is the danger that unless cells are specifically 'locked' or 'protected', then the data held in those cells could be altered or modified thus invalidating the database application. The problem would not arise if the spreadsheet system was not compromised by the database extensions.

The desire to include features because they seem like a good idea or because they reflect popular trends can be pervasive. Notably, the current tendency for poorly-planned evolutionary patching can manifest itself throughout all levels of a computer system, including that of the operating system or machine architecture. A good example is found in the IBM compatible personal computer systems, which require the PC-DOS (or similar) operating system to perform its functions. The actual operating system itself has undergone a considerable number of changes from its original (version one) form, which in turn was largely derived from an 8-bit microprocessor architecture operating system (CP/M[31]). Consequently, most installations of 'super-workstation' hardware - as typified by current 80386- or 80486-based machines - still assume and accordingly *require* an older operating system technology in order to maintain compatibility with the previous systems. This naturally inhibits the machine functionality to a worrying degree[32]. The majority of computer technology

31: CP/M is a trademark of Digital Research, Inc.

32: Tests were carried out and reported in [Powell 89]. An Intel 80386-based machine running a 'C' program under PC-DOS took 49 seconds to calculate some results, and 35 seconds to output those results. The same machine executing the same (re-compiled) 'C' program under Xenix (a UNIX implementation that utilises the 80386 resources to the full) took 16 seconds for the calculations and 4 seconds to output the results. These figures represent 306% and 875% improvements respectively when the machine architecture is less constrained by the operating system.

architectures, which are perfectly empowered to support multiple users and execute multiple tasks, are thus restricted to single-user, single-tasking operation in order to provide application porting facilitation favoured by implementors, but spurned by end-users. This is clearly demonstrated in the Microsoft OS/2 operating system, which on the one hand promotes multi-tasking and a more consistent and considered machine architecture, yet at the same time seeks to maintain a high degree of compatibility with the essentially contradictory PC-DOS architecture.

Section 4.5: Focused Systems and User Confusion.

The research into groupware systems has arisen from the need to look into the communication and sharing issues in more detail. Engelbart's conclusions [Engelbart 88] concerning the issues required for successful implementation of Computer Supported Cooperative Work can be summarised as follows:

1) There should be a common, coordinated set of principles over the application areas. This particularly applies to the existence of a common style of communication.

2) Grades of user proficiency should be recognised, as even expert users will be novices in less frequently used domains. As proficiency increases, users gain access to richer tool environments.

3) There should be easier communication between domains, as well as easier expansion of the domains. It should be possible to move and communicate information between domains easily, and now tools should be installed as required.

4) The system should be easily configurable by the user, adding new tools or extending environments to suit their needs.

5) Support services will be required, for such tasks as design and administration, training and cataloguing.

6) There must be recognition of standards for information interchange and ranges of hardware. It should not be necessary to presume a particular machine or environment. Information exchange and representation should be possible across system environments.

7) Methodologies will be required to augment the knowledge workers' community, as CSCW technology alone contributes only to the tool system.

8) There must be reconsideration of the roles and organisational structures within groups, which necessarily have the potential for change with each introduction of new technology.

As an example of the first point, Engelbart cites the standard user interface for Apple Macintosh applications, which are consistent in their common method of interacting. However, the preferences or suitability depend also on circumstance, so that the HyperCard system on the same machines has a different interaction mechanism.

Given the data presentation capability of existing systems, it is clear that the ideas and technology necessary to handle individual information components already exist, but in isolation from each other. In order to build more advanced information systems of the type described in this thesis, it is vital to give consideration to the integration of the components in accordance with the ideas developed during groupware research. Amongst other things, this will directly affect the ability of user to control or 'navigate' the process of data adminstration and presentation. In order to increase the potential of the system, it must be possible to ensure that the components of the system can be as closely associated (or integrated) as possible, since it is unlikely that any one system will supply sufficient tools to deal with all eventualities. Associating the tools will mean that it will not be necessary for any one tool to try and do all the potential tasks, and instead to concentrate on the specific job that can be done well - at maximum efficiency and speed. The more integrated a set of system tools are, the less easy it is for a user to become confused through having to learn a new set of techniques for using each tool.

Additionally, an extension to the processing model providing integration of resources means that positive and negative feedback properties present within most systems can be modelled, reflecting more accurately the actual circumstances of the tasks. Feedback can be defined in computer terms as the ability of a machine to use the results of its own performance as self-regulating information and so to adjust itself as part of an on-going process[33]. Certain authors go further in stating the importance of the feedback property, with Weiner:

'[regarding] it as an essential characteristic of mind and of life.' [Weiner 50]

Going further, he stated that:

'It is my thesis that the physical functioning of the living individual and the operation of some of the new communications machines[34] are precisely parallel in their analogous attempts to control entropy through feedback.'

The need for modelling feedback within a system is justified in terms of accuracy and efficiency of operation, combined with minimising redundancy and danger of error. Supporting the mind in its tasks of processing real-world data and building realistic models is a responsibility that must not be limited by the use of restricted (incomplete) information systems.

Ultimately, the problem resolves to one of determining the control that the user has during information handling tasks. At present, the control facilities supplied to the

33: Homeostasis.

34: He is referring to the first computers. One of the most famous early American computers, UNIVAC, was operational at the time.

user are generally hierarchical in nature and limited in their range, usually with regard to the options available to the user at each stage of a given task. Although a large number of data handling options may be supplied, the actual manipulation of that data, the number of ways in which the data may be interpreted, can all be strictly limited. For example, the way in which a data file is represented varies according to the type of application which is using the data file. Word Perfect, WordStar and EDLIN all handle text to a greater or lesser extent. Yet the default storage method for each of these text processors is different, resulting in a basic incompatibility between files that are supposed to be of textual form. As a result, although the user is permitted to edit text information with each of the programs, once editing has occurred with a given tool, it is not such a simple matter to exchange the results of the editing with one of the other programs.

This means that control over the data currently remains in the province of the focused application program. The user is 'permitted' to ask the application program to perform a task. Until such time as applications start to facilitate considered data exchange and support recognition of the working environment of the data, it will be impossible to construct true information systems where the control of the system resides with the user. This statement is supported by Benyon et al [Benyon 87], who pointed out that:

> '..we feel that there are still several areas of information systems activity which await adequate modelling.'

Their reasons for stating this dealt with several areas, including the modelling of the interface between systems and user, where:

> 'Defining the data or information which has to flow between systems is necessary for a successful design, but not sufficient. ... The designer often has a bewildering range of options along a continuum, all with different technical requirement and implications for the system's users. ... In most instances, the interface is determined by the designer's stereotype of the user or likely users.'

Such is the limitation on integration of current systems that there is still hesitancy in recommending the use of products labelled as 'integrated office systems'. This is particularly the case for multi-worker environments, and typically the advice is:

> 'Unless the study team [evaluating the proposed system] finds a product that is both excellent today and likely to become even more in the future, it would probably be well advised to wait until the market situation clears somewhat. Instead, it can move forward to discrete services such as electronic mail and add more functionality later.' [Panko 88]

The reasons for this reluctance are straightforward to identify. As Panko goes on to point out:

> '[Integrated Office Systems] fail to provide many needed functions ... lashing IOSs from different vendors together with electronic mail standards alone may not be satisfactory.'

The systems which do exist are extremely proprietary, typifying the problem of focused application systems. However, it is not merely the application that will be focused, but the products themselves will be deliberately distinguished, re-enforcing the problems of integration. Even in the established application of electronic mail, Panko states that:

> 'The existence of a multivendor electronic messaging system environment should be completely hidden from users if possible. In most multivendor environments today, it is *extremely* obvious when a user crosses vendor boundaries.'

The required direction for integration is to eliminate the boundaries that exist between applications. Panko concludes that:

> 'Second generation integrated office systems will end this artificial separation by providing integrated access to most or all End-User computing tools. At first, this integration is likely to be limited to putting all of these functions under a menu and offering file conversion among selected tools, and even this limited integration will be valuable. In the longer term, second-generation integrated office systems should be able to provide seamless integration by offering a common set of user interface features, even if the underlying programs are diverse and, when used in stand-alone mode, offer different use interfaces.'

Paraphrasing, the current state of application development technology is probably barely adequate for most purposes. Panko's ideas suggest that what is required is an operating environment - an architecture - that will support the integration of all the interacting components, including the user.

Section 4.6: Conclusion.

The integration of applications within systems seems to be accepted as being of benefit. However, the modelling required for the development of a genuinely useful information processing support tool requires rather more than flexible consideration of the construction of data manipulation systems, beyond a direction towards enhancing communication of data. A more general model for representing data is required, where its context and environment must also be taken in to account.

This means that consideration must be given to much more than merely computerising certain techniques. Further thought must be applied to establish more useful conclusions, and it is this issue that must be addressed next.

Chapter 5.

Requirements for Genuine Information Systems: A Fresh Consideration.

Section 5.1: Introduction.

A major question that follows from the analysis of current systems is that of identifying the attributes that computer-based information systems should have. In other words, what features are minimally required in order that the application can be categorised as a computerised information system.

This is an issue which many researchers are studying in detail, and accordingly this research has a different primary goal. We are not attempting a formal identification or definition of exactly what a computerised information system is. Instead, we are particularly interested in classifying and supporting the architectural requirements that any given information system may require. Such an approach falls between the two more usual strategies of:

1) Identifying or specifying the problem and then providing the tools for solving it.

2) Providing general purpose tools that can be applied as suitable utilities for solving any/all problems.

This latter approach often manifests itself in the construction of a prototype system which can then be used for experimentation. In many cases, the prototype is extended to become the end product, rather than being discarded in favour of an improved system that is constructed in the light of results obtained from the original prototype.

Typically, developers intending to produce some form of information processing system take as their starting point a study of the existing and more-easily identifiable attributes of paper-based information system, and use those to build the initial specification for the electronic system. Such an approach has its merits - such as simplicity and general awareness, but has the inherent danger of restraining the scope of the resulting applications, precisely because they may unwittingly incorporate certain of the limitations imposed by paper-based information handling. Some restrictions can be quickly identified and eliminated, for example the quantity of information that can be held on a sheet of paper, or the ease with which information can be updated and manipulated. However, other weaknesses are not so easily dealt with. Paper-based systems have no way of representing links that may exist between information on separate sheets, other than the straightforward but abstract notions of relationship via adjacency, reference or footnote. In the former case, sheets close (or next) to each other probably have a closer correlation than sheets further apart - the association normally being that one sheet follows on after the other when read in sequence[35].

Researchers have quickly realised that computer-based systems offer the potential for many more links, of many more kinds, and with explicit detail as to the nature of the link. However, ordinary information users are not yet acquainted with how to use information linkages in this fashion - they are unable to make use of such linkages because the possibility for such understandable linkages simply does not exist outside the computer system, it is not an instinctively understood concept. Of course, it may be possible to learn how to use such facilities[36], and as understanding is gained, so the user would be able to experiment with such linkages, and so the system would become of greater value. But the learning itself is an abstract example of a focused application, in that the skills learnt in using any new system which has some unique features (such as the manner of establishing new kinds of linkages between information items) may be usable only on that one package. Other systems may offer similar features, but which are implemented in different fashions, or with a different philosophy behind them.

35: Although this itself is a result of the limit on the amount of information that can be held on any one sheet.

36: An issue which the prototype architecture described later is intended to test.

Accordingly, there is reason to conclude that using paper-based technology as a prototype model for a computer-based system may result in a product which is digressional or even divergent from the end-goal, because it is inherently focused rather than diverse. The earlier quote by Negroponte[37] gave support to the role of the computer-based system in helping with the presentation of data to the user. But it also has a more subtle implication concerning the nature of presentation of materials to the reader. Determining how the reader would like to have materials presented is not a matter to be taken lightly, as it may directly influence the way in which the data that is presented can be assimilated and interpreted (or understood) to provide information value.

A further point concerns flexibility of systems. Having totally flexible systems, which can be completely controlled by the user, may not necessarily be a good thing. Indeed, it is easy to envisage circumstance where it could be counter-productive, such as when end-users are so confused by the number of command options available at various stages in the package that they subconsciously reject it by refusing to learn or use the system. The question of whether flexibility is a good thing is an important one that will be returned to in section 6.1.

For the moment, it is becoming clear that paper-based technology is a clearly defined and implemented system. It has been rigorously tested and proven, and as such is a completed system. It has no real prospect of further development, because although the underlying technology may change or improve (from quill pens through to fountain pens, pencils and ball-point pens, and currently the 'state-of-the-art' laser printers), the manner in which the technology is used is almost completely unchanged. To try and extend the system is in effect to simply change (update) the technology, not the manner of working - and trying such an extension is unlikely to have measurably beneficial results. Indeed, it is important to consider whether we really want to extend the system. The next stage would almost certainly be computerised systems, but following this path may not be desirable in practice.

37: 'electronic displays (should) adaptively display not only what materials readers wish to see, but also how they would like it presented.' [Negroponte 76]

An alternative approach is required. With computer technology, we have a tool which can perform a multitude of tasks according to the instructions we give. The difficulty therefore is to identify what tasks we wish the computer to perform, and thus we must identify what we require from an information system. The idea of computerising paper should now be discarded in favour of a more considered recognition process. We must move to consider what tasks are actually required during all the stages of data handling by us, the human users.

Carrying out an examination of the various systems used for handling data will enable us to identify some of the key components which are involved in performing the various tasks desired by end users, and for which Information Technology systems can make a valuable contribution. Many of the systems described in chapters 2 and 3 deal with the components to an extent, but none of them deal with all the components as a group. This thesis therefore involves in part the development of an architecture that permits a wide range of application activities.

There is an important proviso to consider when dealing with generalised information systems, however. There is the possibility that such universal designs cannot be defined in a machine-independent fashion because of the limitations of existing hardware architectures. Non-dedicated systems may be constrained by the host operating system and/or hardware, while dedicated systems may be implicitly focused. Consideration must therefore be given to the question of whether this implies that the validity and efficiency of a computer-based information system depends directly upon the hosting machine or software architecture.

In other words, we must consider the possibility that any computer-based information system will be directly and intrinsically affected by the technology that supports it, whether it be hardware or software. If it turns out that this *is* the case, that the system *is* directly influenced by its environment, then we must consider to what extent is it true that a reconfiguration of the architecture is a vital and mandatory task that must be performed in order to orientate the information system towards another application.

Section 5.2: Identifying System Components.

In chapters 2 and 3, we described some examples of commercial and research systems which have typical components identifying some of the processing tasks that should be provided in most data handling systems. We can examine the components to help with the derivation of an interaction diagram that portrays the inter-relationships of the common tasks that could be carried out during information processing. The diagram details how these component tasks contribute to the overall system by specifying how they work and interact with each other. This approach of exploratory system development using characteristic identification has the advantage over other approaches in that we are working towards a second generation model of information system attributes, based on the first generation approximations[38]. At the same time, we are attempting to eliminate pre-conceived ideas which may be dependent on constraining suppositions. Initially, however, we will consider the major system components as they are currently implemented or viewed, before reviewing how they can be consolidated as required for the more enhanced information system and later the information environment.

Of particular interest is to consider the normal working environment of each of the identified components as depicted in Figure 4. As a top-down sequence is followed, it is obvious that the components become more technical but lower-level in their nature. Equally, however, the provision of the component within a specific application package becomes less probable; instead more of the functions would be found within the standard operating system resources of the supporting computer system. This dependency on the hosting architecture will clearly influence the nature of the component and the nature of its operation within the application. This leads in turn to the suspicion that an enhanced information system cannot be defined in completely machine-independent terms. If this is so, then any application or use of an information system will also have a corresponding optimal implementation of the system itself. Otherwise, the application will in the least case be inefficient, and in the

38: First generation models for information systems are those which have a clearly defined set of tasks devoted to the handling of a limited set of data that purports to represent information. Such models are inevitably focused application systems.

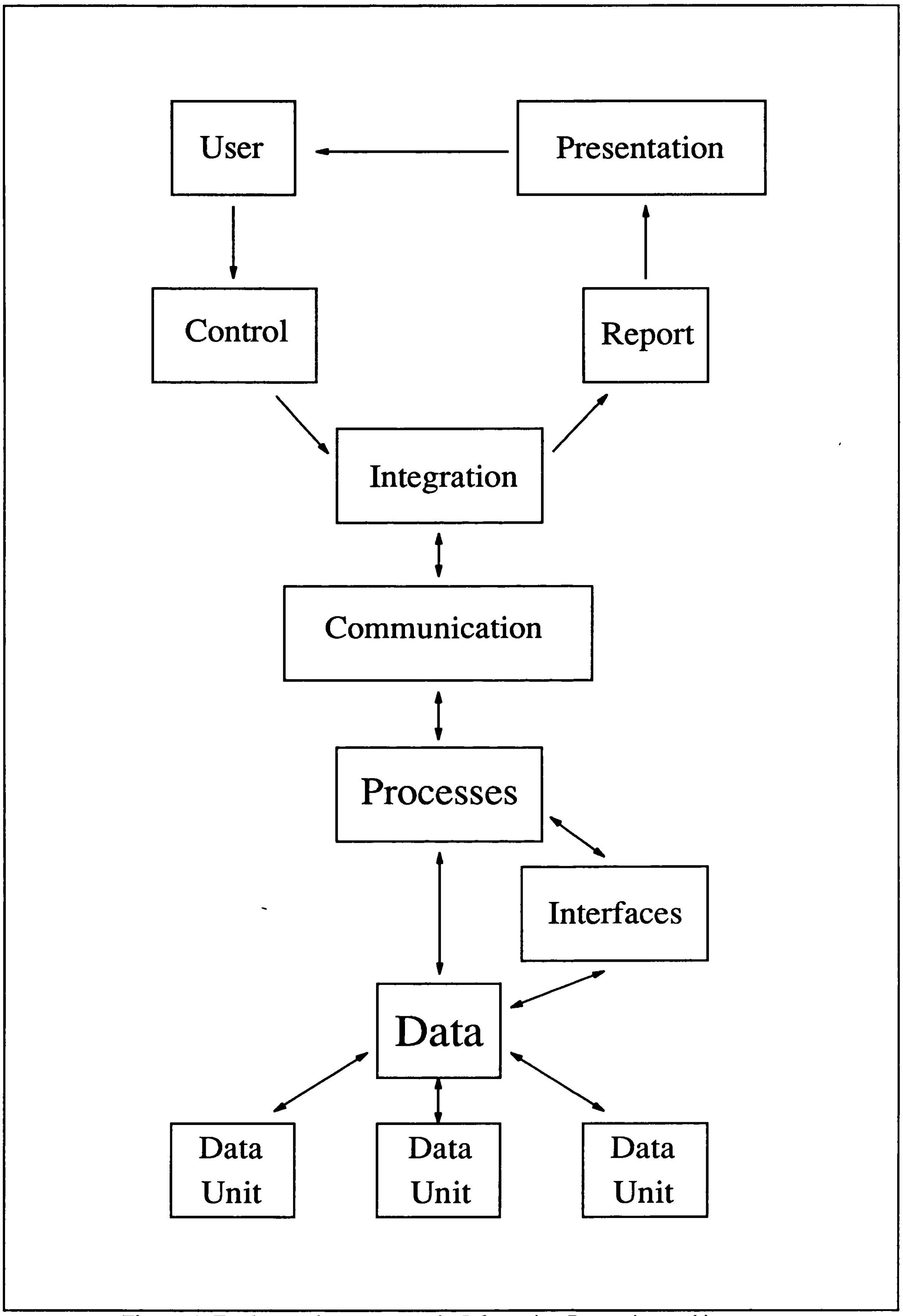


Figure 4: Fundamental components in Information Processing architectures.

worst case will simply not function at all. The process of using an information system for another task[39] would therefore imply that a reconfiguration of the system components must occur - in other words the information system architecture must be reorganised. Thus the implementation of the architecture would have to be inherently reconfigurable, almost to the point of amorphous existence. This naturally introduces enormous difficulties regarding control and data flow, which are specific instances of classic problems that must be dealt with by operating systems: namely multi-tasking and sharing of resources without deadlock. The discussion in section 3.7 of the work by Randell is relevant in this context.

Section 5.2.1: The User and Presentation to the User.

This is the top-most level of the information system, and as such has the most direct relevance to the end-user. However, for most applications packages, the top-most level is defined as being exclusively devoted to presentation of data to the user. A considerable amount of work is being done in the field of Human-Computer Interaction, and it is at this level that the results obtained are applied. However, for most applications, an actual recognition of who the user is or what they might require is casually dismissed, in preference to the 'standard' user. In practice, the established model of a human operator seems to bear as much relevance to the real-world end-user as regular shoe sizes bear to human feet. Certain customisation may be offered through changing specific application attributes. The range of variables offered typically fall into three categories:

1) How much help is wanted, on a scale ranging from minimal up to maximum.

2) What colour scheme would be preferred, and occasionally how many lines on the screen.

3) What key-strokes should be used to carry out the ordained tasks.

39: Sometimes called 'launching' an application.

In practice, the provision of help is at best a screen-based copy of text that is already provided within the user manual. Although there are some instances of packages supporting the idea of truly context-sensitive help facilities, it is inevitable that such features require significant processing by the application in order to establish what the difficulty is, or what the user wants help with. As a result, system designers are somewhat reluctant to devote time and energy to a feature that would - by definition - reduce the amount of computer resources available for the tasks of the main application. Certain computer architectures can help by supporting the basic idea of switching between tasks or sections of code[40], but this will often take a noticeable amount of time and so is deemed unrewarding effort.

Clearly, there is a need for a support mechanism recognising individual characteristics of multiple users. This concept needs to be incorporated into the working model of the information architecture, although the degree to which the incorporation of support is necessary would be influenced by the adaptability of the users. Aside from the obvious issue of ensuring that the user can actually operate the system, the user is also the upper-most command level for the entire information handling process, endowed with considerable decision making skills as well as the ability to make intuitive selections on the basis of meagre data. Moreover, much computer-based information handling assumes that only a single user is involved in giving direction to the processing. As discussed earlier in section 3.8, a few notable exceptions (under the category of 'groupware') attempt to offer working environment that permits multiple users to cooperate in information handling, although in practice their functions seem to be directed exclusively to sharing access to a common diary and provision of electronic mail. Currently, multiple users are unable to work concurrently with each other except within very specific and restricted circumstances. In order to achieve a reasonable degree of efficiency, all possible processing components - including multiple human users - must be modelled or at least represented within the information architecture so that communication and cooperation can be supported, and feedback included. This is a point which will be discussed in section 6.2.

40: Called 'swapping' on truly multi-tasking machines such as UNIX computers, or by the use of 'overlays' on single-tasking machines such as IBM PC compatible processors.

Section 5.2.2: Control and Report.

At this level of information processing, directional commands are being passed down to the main application control section, and results or reports are being passed back up to the user(s). Here, the modularity of application systems means that it is possible for many packages to isolate clearly the tasks from the main application code, thus enabling the user interface sections to operate independently of the underlying application. Indeed, there are certain examples of packages where this is considered a bonus[41], as the code can be re-used with minimal modification, enabling a 'similar look-and-feel' to be promoted, and so reducing the apparent learning time before the package can be understood. Equally, the module segregation means that extensions can be more easily accommodated, for example to allow a mouse to be used (often for exactly the same tasks as are already performed by the cursor keys).

However, distinguishing between modules and incorporating two distinct and unidirectional paths between the user and the internal system means that the information passing along the paths must necessarily be formatted in a variety of fashions, suitable for passage along the differing communication routes. Thus for the control component, this will probably entail parsing the user's instructions into a known format - defined by the system designers, and often at an early stage of the design process - that can be accepted by the internal modules, which is a perfectly reasonable mode of operation.

Nevertheless, when this same technique is applied to data passing from the internal system to the user presentation components, then there are problems concerning the need for re-formatting according to the presentation device(s). In effect, this is a limitation imposed by the design which concerns the range of data representation styles, as well as introducing the danger of losing data context. This is an issue that will be addressed in section 6.2.

41: For example, the 'Turbo' family of compilers from Borland International, which have very similar editing environments and control mechanisms.

With regard to the control aspects, there are certain circumstances in which control should not be as absolute as is commonplace at present. Obvious examples would include requesting confirmation before a user is permitted to delete large groups of files, or to re-format a disk that has data on it. The user has absolute freedom to instruct - thus controlling - the system to perform an erasure. Despite the confirmation check, users quickly become immune to the request and will reply in the affirmative automatically. Normally, this gives the desired result, but in the single exception will result in devastating consequences. The blind obedience of a system in executing a command must be tempered by interpretation of the command within the circumstance of the actual event.

Taking an analogy from the motor-car industry, a driver may instruct the vehicle to stop quickly (by pressing hard on the brake), but to do so may cause the wheels to lock causing the vehicle to skid (complete loss of control over vehicle). The solution to this is to have advanced braking features fitted, so that the car can 'measure' if a wheel is about to lock, and automatically release the brake pressure a fraction until the threat of wheel lock is over. In other words, the command to 'stop the car' is obeyed by applying the brakes, but the environment of the command is also taken into account so that if obeying the command literally could cause additional problems, then the interpretation of the command can be relaxed slightly.

Control of information processing must be made context-sensitive in all its aspects[42]. In the same way as inexperienced users can do a lot of damage through lack of knowledge, so experienced users can do a lot of damage through overconfidence. A command affecting data items may directly influence past, present or future work of other users, and so must be evaluated in the light of this. Obvious examples are addressed by normal security measures, including the access rights to files (who can delete, read or write to them), or who has access to resources.

42: As with all things, there are exceptions. The Paris 'Airbus' disaster on June 27th, 1988 [Times 88] was a 'no-win' situation for the system: If the aircraft attempted to climb too steeply, it would stall and crash, so the control system rejected the pilot's instruction to climb sharply. But by rejecting the command, the aircraft system prohibited any possibility of avoiding the ensuing crash into nearby woods. The system behaved correctly, but the pilot was found to be at fault [Times 88b] in ignoring warnings of being too low.

This inevitably leads on to the matter of reporting the results of commands. Clearly, the initiator of a command should be advised of the result - whatever it may be - but there are instances when other users should be notified. The system/security manager should be alerted if a security transgression has been attempted or completed, and co-workers should be notified of modifications to any data which they are using in their work.

Much of the research into Human-Computer Integration is directed at this particular level of information processing, and the results include such concepts as the WIMP environment (Windows, Icons, Mouse, Pull-Down menus), and this technology has been gratefully adopted in other systems, such as hypertext packages.

Section 5.2.3: Integration.

At this level of system operation, data exchange can occur. Specifically, two distinct forms of integration can be implemented within application packages. Firstly, control instructions from the user (or, in certain cases, from external events) are combined with existing status information to identify the appropriate tasks that must be obeyed, within the context of the environment. Secondly, an integration can be offered whereby control or result data can be transmitted to or received from external modules, which exist outside the scope of the application. This latter form of integration is comparatively primitive but more widely implemented, and where it is offered will normally support only clearly defined (and often limited) standard systems. Typically, the implementation is found in either a dedicated and cooperative set of application tools, or else in some form of distributed system. In each of these cases, it is important to recognise that the underlying task is to transmit the required data in the desired form. The selectivity of access to data and the need for appropriate methods of presentation of results are both issues that are relevant to database systems. Accordingly, much of the technology that has been developed for supporting such data organisation and retrieval in information systems has been derived directly from existing database management system (DBMS) technology.

In this case, however, we are particularly interested in expanding the implementation of integration to work with all data types, and to retain the benefits of context sensitivity and relationships between data, wherever this is possible. This requires that there be some form of manipulation model that can operate with typeless data, specifically to avoid problems of proprietary architecture in any form. Unfortunately, typeless data is a rather low-level entity, and can suffer from the inherent dangers of inadequate specification of context. As a result, this may lead to data being interpreted in a meaningless or even erroneous fashion. However, as will be seen in section 7.2, we can deal with this difficulty by the incorporation of environmental attributes which will be used - in the minimal case - to provide state associations to the typeless data. The advantage of such an approach is that by distinguishing between the characteristic attributes of data, of relationships, and of the environment; we can develop application models that are equipped to support interpretation of the data in terms of its purpose rather than exclusively on the basis of what type it possesses. The elimination of any dependency on implied type definition means not only that we can reduce[43] the likelihood of producing focused information systems, but furthermore that we can reduce problems that may be caused by the system being dependant upon specific host technology architectures. It must be re-emphasised that we are not excluding the attribute of 'type-ness' for a data item, merely distinguishing clearly between the two. For genuine integration to occur in an enhanced information processing architecture, it seems necessary firstly to recognise that there is a distinction between a datum and its attributes, and secondly to provide a means for communicating the two associated components.

Section 5.2.4: Communication.

Although this component is dealt with as a middle-level component within the interaction diagram, it has influence over all of the components within the interaction diagram because it directly affects the movement of data and attributes (as described in section 5.2.3), and also the flow of control throughout the entire information processing architecture. In its elemental form, communication is concerned with the

43: Ideally, exclude.

movement of objects between object users, whether they be computers or consumers[44]. Considerable expertise and technology exists to handle the material tasks for fundamental data communication. But including the concepts derived at the higher levels of the interaction diagram means that the necessary communication techniques must become more abstract in nature, as they also depend upon the method of portrayal rather than just the mechanism of transport.

Despite the large quantity of research that is carried out concerning communication of data within and between computer systems, it is a notable fact that much of the work is carried out with little regard to the nature of the data itself. Typically, researchers will refer to packets or 'lumps' of data being carried around the messaging system, and this proves quite sufficient for the advanced implementation of powerful systems. The actual interpretation of the data is (correctly) conditional upon the objectives and operations of the initiating and receiving applications. The result of this is that some extremely powerful mechanisms exists for conveying data, but in many cases the client applications that use the conveyancing tools are much less developed or advanced. This has resulted in a slow acceptance of communication technology, other than at rather low or elementary levels where the applications are straightforward and unquestionably beneficial[45]. However, there is little doubt that an enhanced information processing system must expand upon the fundamental data communications technologies to support the wider requirements of context, attribute and relationship communication along with the data itself.

Enhanced systems will inevitably require a portability of data between the multiple applications within the system, and this is achieved in most cases by having a standard data representation format. Hypertext systems will implement communications in only limited ways, as there is in fact rarely any requirement to share data, only to be able to follow the links that can be established between the data

44: Computer in this context meaning some form of processor - not just technology-based - and consumer meaning a data source or destination.

45: Data exchange between incompatible machine architectures, message passing across long distances (inter-machine), and memorandum-based dialogues. This latter example may not necessarily be beneficial, but is a social rather than technological issue, and so will be not be discussed further here.

items. Indeed, Oren has pointed out in [Oren 88] that such requirements make CD-ROM an ideal technology for hypertext systems, because:

> 'The fact that CD-ROM is static (i.e. it cannot be altered) is a selling point, because large editable hypertext systems can be difficult to update. Every time you edit, ... links connected to that node may have to be changed. Although suitable data structures do this, it can place quite a computational load on your computer if you are dealing with a database consisting of hundreds or thousands of nodes.' [Oren 88]

Section 5.2.5: Processing and Computation.

The processing and computational tasks performed by most information handling tools are characteristic of focused application systems. As described in section 5.2.3, the constituent data has been stripped of all but the most fundamental (scalar) qualities. Context has been removed, or at best deferred. The application tool can now proceed to manipulate in accordance with the direction of its design goal, which is inevitably focused. Almost without exception, the nature of the activities performed in carrying out any data processing within an information handling system will reflect the fact that the *information system*, and not the user, imposes the pattern and manner of operation. Thus, the implementation of mechanisms for processing are a result of the nature of the permitted functions; whereas it would be preferable to have generalised tools for use across multiple applications and functional environments.

This is clearly demonstrated in so-called integrated systems, where the multiple tools require the ability to share the data. This can only be done if the data storage is in a known and agreed format, and this can often impose limitations on the ways in which it can be accessed or communicated. Consequently, the tools which can be used for this exchange have to be limited, and so constrain the application. A simple example of the problem can be seen concerning the use of different operating systems on otherwise identical machines, which may sometimes disagree on the representation of apparently standard data types.

For example, a text-file on a UNIX system has the <LF> character (ASCII code decimal 10) as the delimiter for the end of line, while a similar file on a personal computer using DOS will store the file with the end of line delimited by the sequence <CR> and <LF> (ASCII code decimal 13 then 10). This subtle difference can cause problems if the same data is to be used by separate processing tools, a possibility made more likely with the introduction of UNIX machines into the PC working environment[46].

In a similar way, the representation of other data types can cause problems, for example the representation of integer or floating point values. Taking the example of UNIX again, there are invariably at least three forms of integer available in the C programming environment, namely `SHORT INT`, `INT`, and `LONG INT`. The only guarantee about their representation is that `SHORT INT` can hold a range of integers that is less than or equal to that of `INT`, while `LONG INT` can hold a range of integers that is larger than or equal to `INT`.

Non-computational devices are not immune either. The method for representing a graphic line on an output device can vary dramatically between (say) printers, raster-scan or vector-scan monitors. Accordingly, appropriate processing to produce the required graphic line is not simply a task of computing the end-points, but recognising the output environment.

Section 5.2.6: Data Groups.

The collection of data into groups is a complicated issue, that is further restricted by the design of the hosting machine and operating system architecture. Files may be stored in a variety of fashions according to their type, while certain environments will simply store data in a byte-wise fashion[47]. Considering the matter

46: In order to help introduce UNIX into the PC market, certain manufacturers offer DOS emulator tools that run under UNIX. The emulators mimic standard PC with graphics, memory, disks and (normally) an 8086 processor.

47: Under UNIX, a byte is not so simple a concept. What is the difference between a byte and a character? Signed and Unsigned? Some systems store characters as short integers, or even full integers.

from a higher level, hypertext systems will normally work with a basic unit of a page or node, which can actually contain several kinds of data: text, graphics, references, 'buttons', etc. The representation of these complex data groups is a major problem for the portability of information across systems and architectures. Yet this same grouping ability is desirable because it enables us to work with more complex problems, by allowing us to categorise and then associate or relate the possible sub-divisions of the more detailed and higher-level task. By working with a group rather than a single entity, we can apply a particular technique to a potentially complex object which is represented in a much simpler working form. This has the advantage that the otherwise complicating issue of extensive detail is less likely to obscure the approach used to solve the problem.

Section 5.2.7: Data Basics.

It is very difficult for computer systems to agree on even the simplest concept: the bit. Bits being transmitted across a communications link can be represented in a number of forms: sometimes by varying voltages, sometimes by phase changes in the carrier waveform. Storage on a disk can vary also, with GCR (Group Code Recording) being completely incompatible with MFM (Modified Frequency Modulation).

Even assuming that it is possible to agree on the storage of bits, other problems arise concerning what the values represent or even with regard to the order in which they should appear. Certain microprocessors require data to be stored or accessed in high-byte/low-byte sequence, while other processors require the opposite sequence. Any given data value can be interpreted in a variety of ways. A single 8-bit byte can represent 256 possible combinations - but of what type? Different colours, voltage levels from an analogue device, speed of transmission, etc.

This confusion over potential meanings results in difficulties for integrated systems, which must necessarily include some agreed way of determining what data values represent - attributes must be defined. Unless the attribute mechanism itself is agreed upon, then data exchange between integrated systems is an impossibility. Furthermore, the representation of attributes normally occurs at a very low-level, with close association to the data; while in fact there may be circumstances where a higher-level attribute concept may be preferable.

Section 5.3: Suggested Requirements.

Having identified the components that appear as tasks within information processing domains, we must now collect the elements into an integrated architecture which will represent the structure for a possible second generation information system. From the interaction diagram of section 5.2, it is clear that there is a path from the lowest levels of data representation through to the highest level of user presentation and awareness. Accordingly, it is important to ensure that the data representation chosen can be used throughout the whole of the interaction model. The data handling itself would then be built upon the standardised representation. For the inevitable circumstances where the representation is not adequate, then either the model must be enhanced or an automatic (transparent) conversion facility must be offered. The former option is preferable, not least because it promotes the cause of genuine integration of processing components, although it initially requires more effort to achieve.

As we move down through the interaction diagram, so our understanding of the components becomes more thorough. Indeed, it can be argued that each of the components represents historical stages during the development of information systems - especially those which are computer-based. The problem is that this promotes an imbalance with regard to the quality and scope of understanding, as well as an inevitable bias towards lower-level system development on the part of system designers.

Nevertheless, our objective must involve an attempt to understand in practical terms what is required in order to support the mind that is processing information. Furthermore, as discussed in section 6.1, there is no reason to suppose that there should necessarily be any precise correspondence between the processing within the mind, and the processing within a computer-based information system. As Roszak has indicated [Roszak 86]:

'there is a vital distinction between what machines do when they process information and what minds do when they think...that distinction needs to be kept plainly in view.'

In order to understand the goal of our desired information system, we require a working definition of Information. An uncomplicated explanation was provided by Roszak [Roszak 86]:

'information ... is ... discrete little bundles of fact, sometimes useful, sometimes trivial, and never the substance of thought.'

Such an interpretation, unfortunately, lacks the detail or precision that we require for developing our working model. However, it does promote clarity, which is to be desired in preference to complexity or obscurity. Accordingly, for our model, we define information as being that which is gained by the observer's interpretation of data presented within a dynamic environment. The term data refers to the raw material used in all kinds of computation and inference, whether by man or machine, and regardless of source or destination. The term observer refers to the end-recipient of one or more results, or alternatively to the supplier of data. Typically, this will be the mind of a human user, but it could equally refer to a data storage mechanism. The environment is a concept which is discussed at length in section 6.2, but taken in conjunction with the dynamic nature of changing circumstances, is vital to incorporate feedback within the system. The observer's interpretation of data is the key to understanding information in terms of computer-based information systems. data, no matter how presented, is meaningless until interpreted by an observer. Since the experience of the observer will directly affect how he or she considers the data presented, the interpretation will influence the information derived from the data. For example, most people can interpret pictorial data more easily than numerical tabular data. Accordingly, we can examine each of the defined components in the light of our interaction diagram to establish a model of the required information system architecture.

This concept of interpretation seems to be supported by Roszak, although he gives it a different name of generalisation. He first states as a paramount truth:

'the mind thinks with ideas, not with information.'

Using this, he suggests that:

'The relationship of ideas to information is ... [called] ... a generalisation. ... Generalisation is the process of creating or seeking for a sensible, connecting pattern; possibly by enlarging upon a very few facts to produce a conclusion. After more facts are gathered, the pattern may fall apart or yield to another, more convincing possibility.'

Elsewhere, he states that:

'The relationship of ideas to facts are that ideas are the integrating patterns.'

The direction of these statements is that in order to function, the mind must have access to facts. These facts are then grouped, collected and arranged into patterns which relate the facts. The patterns themselves form ideas, which is what the mind thinks (works) with. An information system which purports to support the mind in its thinking (processing) would therefore have to assist the mind in accessing facts, and subsequently provide the facilities for experimenting with relationships and subsequently arranging facts into patterns.

Facts, or data, is the collection of raw material worked with. The great difficulty with data is that understanding what it is can be intuitively obvious, but hard to formalise. However, in computing terms, we can simply define it as being fixed or altering values of variable type, suitable for processing. The observer can similarly be defined as a human user of the information system.

This means that we have two further aspects to work with in order for our model to become an information system: presentation and the environment. Both of these issues must therefore be addressed and supported by the architecture of our system. Indeed, in section 7.2 it was shown that both of these issues can be unified by modelling the entire environment within a suitable architectural structure.

The issue of presentation of the data to the observer is an important one, as implied by its relevance to the tasks of interpretation. The problem lies in determining what form of presentation is the correct (or preferable) manner for a particular set of data. For some data types, this is a comparatively simple matter: sound can only be presented in one way, with essentially uncomplicated variables concerning tone, volume or speed.

If alternative variables are introduced, or permitted variables are forced outside 'reasonable' limits, then the result will normally be data of zero value, because it becomes (literally) noise. Equally, sensible lighting or temperature control decisions fall within the range band of 'minimal' through to 'optimal', but can eventually be extended through to 'too much' or reduced to 'too little'[48].

However, certain other data types are not so easily fitted into a uniform presentation scheme. Numeric data, for example, can be extremely precise, but taken in quantity can 'blur' to the point of becoming a meaningless jumble when perceived by the observer (hence large tables of numbers are confusing), whereas redrawing the numeric data in the form of a suitable diagram[49] should result in a much clearer display - as long as it is not deliberately constructed to confuse or misguide - although with reduced precision. It is a recognised fact that visual presentation of data to humans has a higher priority than almost any other form of data acquisition through the five senses, not least because of the vastly greater rate of data transfer and recognition that is possible. Accordingly, visual presentation is by default a primary component of any computer-based information system. Consider the tasks of depicting petrol consumption over a period of time to a car owner. There are three forms of presentation of the data alone:

1) Raw tables of numeric data.
2) Graphical representation only (using patterns to depict the differences or relationships between the values).

48: Exactly what constitutes 'too much', or even 'none', will depend on the observer, but there is little doubt that the observer will intuitively be aware of how much is too much.

49: Deferring the problem of what exactly constitutes 'suitable'.

3) Labelled graphs, which combine the graphical representation with context (legend) and sample values.

It is probably agreed that the latter form is the more valuable since it has the potential for communicating the data in a clearer manner. Thus we have the requirement for working with not just data, but also the patterns that can be formed with them, in accordance with the work of Roszak described in this section. Thus we require an architecture to support pattern/relationship establishment. This same architecture must go on to offer a suitable presentation mechanism as well.

A significant problem, however, is that because a computer is used as the host for such an information system, it is relatively easy to offer the same data in a variety of visual forms (the numeric or graphical presentation referred to in this section), and so the complexity of this issue is increased.

Equally, however, as a user is trained and gains experience in interpreting data that is presented, so the quantity of data presented can be increased, and the precision of the presentation improved. Thus, we must allow for the fact that any given data item or group could be displayed in a variety of ways, according to the competence of the user. In effect, we must be able to take some data, and submit it to a range of potential transformations, determined by the circumstances of the environment which includes the hosting technology and also the user's abilities. The actual working environment is thus instrumental in dealing with the problems of data presentation, and as such must be taken into account.

Facts grouped into patterns form ideas. Ideas generate information (according to Roszak). Therefore, given the two options of:

1) Data presentation, and
2) Data and Pattern (relationship) presentation,

the second option will give you more information. Examining that is a purpose of generating the proposed enhanced information system architecture.

Section 5.4: Data and Information, Flow and Control.

We now have two domains of interest with regard to the development of a second generation information system model. The first concerns the means of representation of the complete circumstances of data manipulation, from the initial stages of acquisition, through the tasks of control and manipulation which are preparatory to presentation. The second field concerns the inclusion of all the interaction stages of information handling into the architecture, as concluded from the studies of existing systems and research. The derived interaction diagram leads us to deduce that flow of data in various forms will be mandatory, but that associative with that data flow will be control and command directives, tailored and influenced by other operations occurring within the application model of the environment.

Distinguishing between data and information is key to the problem of implementing an architecture, because as we have seen, information is only partly defined in terms of data[50], and unless the data manipulation is complete and sufficient in its context, then information cannot be acquired. Too often, existing systems have confused the problems of data processing with information representation and so have misconstrued simple data value for system validity.

Furthermore, we must develop in parallel a mechanism for specifying the control of data manipulation within the architecture. The data must be managed with respect to its circumstances, and as such the circumstances affect and influence the data; equally, control implies manipulation and alteration of circumstances or status as well as affecting data, and so must also be modelled.

The provision for complete (but sensible) user control is essential in order to reduce the likelihood of the information system architecture becoming focused upon a single or specific set of application tasks. Extending the focus will bring in many intricate problems that are associated with any large scale architectures, but to do so is imperative if fuller integration of all data processing tasks is to be achieved.

50: According to our definition.

Focused application systems impose the pattern or idea (possibly through an imposition of the operational technique). Conversely, an enhanced information system should let the user build and work with their own patterns and ideas, thus giving greater information value.

Accomplishing this will require the derivation of an architecture for modelling the basics of data and directives with the tools of flow and control, influenced by the inherent representation of environmental circumstances. The production of this architecture forms the major thrust of this research.

Chapter 6.

The Information Environment: A Basis for an Enhancement Strategy for Information Systems.

Section 6.1: Introduction.

In previous chapters, we have worked towards establishing the key characteristic components that would be required for any system purporting to be an information system. More accurately, those components which seem to be required as a minimum in order to support an information processing system have been identified. However, it has been established that it is limiting to group only these components, even in an integrated fashion, unless some context is applied to the resulting system so that the scope of its functionality can be increased. In effect, we must move to consider the extension of information system processing to the greater goal of the Information Environment.

It is reasonable to suppose that humans store information to build models of reality so that they can deal with the world. The models will be comprised of components, such as factual images (basic data), or relationships (such as 'mother of ...'). These components are combined to form views of the world. As Roszak has indicated [Roszak 86]:

> 'Information does not create ideas, by itself it does not validate or invalidate them.'

But the components must derive from somewhere, as signals arriving from the environment in which we exist. Thus as the environment changes, so our working model must inevitably alter as well. As Machlup et al have discussed [Machlup 83]:

'Experiences may initiate cognitive processes leading to changes in a person's knowledge. Thus, new knowledge can be acquired without new information being received.'

Putting this in context with the work of Roszak, we can interpret this as meaning that as the mind develops the ideas which model the relationships between data, so those relationships can be used to generate more data. This is an important suggestion concerning the behaviour of humans in processing data, and if it can be accepted as true or at least valid, then we can use the concept in building computer-moderated systems that will attempt to support some of the processing activities of the brain. However, in order to accept the concept as tenable, we must establish whether the use of relationships in generating further data is a defensible technique. In order to do this, we must experiment upon the generated data to verify whether it fits as a genuine member of the original pattern from which the relationship was derived. However, it is difficult to prove that a relationship is valid in all possible cases, because we may not be able to obtain or test all possible cases.

Furthermore, if our ability to receive the data is limited, then our model of reality will inevitably remain restricted. It could not be extended through the reception of additional facts which either support or refute the model. Such a restriction has significant implications for information-using organisations which assume that in order to maintain continuation of success, they must continually remain up-to-date with computer-based information systems technology. Such a view is popular at the present time, even though it is easy to establish examples where automatically upgrading or incorporating the latest systems can be detrimental to an organisation. Translating information as being the essence of perceptual images, Toffler indicates [Toffler 80]:

'New information reaches us and we are forced to revise our image-file continuously at a faster and faster rate. Older images based on past reality must be replaced, for, unless we update them, our actions become divorced from reality and we become progressively less competent. We find it impossible to cope.'

A straightforward interpretation of this statement is that to remain competent, we (and consequently organisations) must have up-to-date information. However, there are equal risks associated with information overload, where excessive data can either overwhelm or confuse the recipient. Nevertheless, an implication of the stated need for accurate information is that in order to progress and develop, a flexible response model must be developed which can receive the maximum scope of data, and respond appropriately. Such a strategy would be used by computer-based information systems to implement a working model of the real world. In effect, the strategy would then reduce the limitations that currently constrain computer-based information systems. Thus, if used correctly, the computer can help with information processing, as indicated by Toffler:

> 'Because it can remember and interrelate large numbers of causal forces, the computer can help us cope with problems, ... sift vast masses of data to find subtle patterns. ... It can even suggest imaginative solutions to certain problems by identifying novel or hitherto unnoticed relationships between people and resources.'

Although this view is somewhat idealistic, it does identify possibly the most valuable 'skill' of the computer, namely its prodigious data storage and scanning capabilities, although the resulting value would inevitably depend directly upon the nature of the scanning facility.

However, given the potential for advantages derived from using the fundamental capabilities, it is suggested that a new computer-based information system model be achieved by working towards the implementation of an architecture which allows for the modelling and refinement of the data and relationships, each forming components that build a model of the information being processed. Certain current systems allow for representation of individual components, But few offer the ability to construct anything other than simplistic models from them. Much of this is due to the lack of ability to construct supplementary relationships and flexible models of the underlying structure, and this in turn is due to two things:

1) No one processing object can build a sufficiently large model to complete a description of anything other than the most elementary information environment or application circumstance.

2) The alternative mechanism of grouping together lesser but cooperating components to build a more powerful system-manipulating 'organism' is currently restrained through the current inability to offer suitably powerful cooperation and communication facilities: in other words the absence of true interaction between the constituent parts.

From all that has been discussed before in section 5.2 and 5.3, it is clearly vital to achieve a genuine consolidation of all components that can exist within an Information System in order to expand the concept into the Information Environment. The combination must be enhanced beyond the current popular conceptions, in order to support information feedback within and with respect to all components of the environment.

In order to improve the operation of computer-based information systems, an architecture would be required that permits and promotes modelling of the entire Information Environment. We can view each of the elements portrayed in the interaction diagram discussed in section 5.2 within the context of general information processing. The three top-most levels - User/Presentation, Control/Report and Integration - are clearly relevant to the objectives of the given information processing application, but should be further refined to include environmental considerations. Concerning integration, this affects the operation of the tools selected and used for performing the necessary tasks, primarily with regard to the scope of the tools. The middle levels responsible for communication and processing are relevant to the tools themselves, because they control the manner in which the tools operate on the data which is received and generated. Finally, the lower-levels of data groups and raw data relate to the materials being worked with during the application activities. Accordingly, any information processing architecture must incorporate all of these components as a minimum in order to offer the potential for enhanced functionality.

An associated comparison may be drawn from more generalised systems research work. Traditional thought has emphasised analysis of the components, often at the expense of the value of context, whereas the systems approach stresses [Ramo 69]:

'a total, rather than fragmentary, look at problems.'

This has lead to a movement towards recognising and incorporating the feedback relationships among both subsystems and the larger wholes formed by these units [Toffler 80]. This is clearly urging a move towards a more consolidated way of looking at problems, an emphasis supported and promoted within many other disciplines [Lopez 73]:

'Environmentalists tend to see things quite differently ... Their instinct is to balance the whole, not to solve a single part.'

There is an inherent danger from attempting to group together smaller components into a cooperative whole, that of chaos or breakdown resulting from runaway fluctuations within the system. In any complex system, from molecules in a liquid to the neurons in a brain or the traffic in a city, the parts of the system are always undergoing small-scale change, they are in constant flux. When negative feedback is applied (through user control), the fluctuations are damped out or suppressed, and the homeostasis or equilibrium is maintained. But if amplifying or positive feedback is at work, the fluctuations may be magnified so as to threaten the equilibrium of the entire system. Such fluctuations may arise from changes in environmental circumstances. However, in the event of breakup of the equilibrium, the result is often the creation of a new structure at an entirely higher level, called a dissipative structure [Nicolis 77].

Accordingly, in the proposed information processing environment, a monitoring process must be built into all contributing components, in order to minimise the difficulties of destructive fluctuation. Given that an Information Environment model would be implemented using a truly multi-tasking architecture, then there is justification in requiring that some system resources should be dedicated to monitoring performance of the system (breakdowns, fluctuations etc.), as well as to oversee security aspects of the system. In essence, the system should attempt to maintain a fail-safe operational state, so that all potential kinds of insufficiency or failure (hardware, software, communication, integrity, secrecy) could be prevented, or repaired.

Section 6.2: The Information Environment.

It is clear from the discussion in section 6.1 that an Information Environment is in some way an extension to the concept of the computer-based information system, but it may not be clear how such an extension may be represented or defined, particularly in terms of computer technology. We can define an 'Information Environment' to be the representation of the complete set of all components - processes and variables - that take part in the entry, manipulation or presentation of any data to any number of users. Constants within the environment effectively define the application under consideration. Hence the users are modelled as part of the environment, as are their workstations, their data, the input and output devices, the co-workers, and so on.

There is no reason why a user should be prevented from achieving any desired task in precisely the way that he or she wishes, and the key objective of developing an Information Environment architecture is to provide the required flexibility. By working with data in groups of known form as the basic units, processing modules existing within the environment can use data in any way desired. Data intended for use in a particular application is stored in a 'global' form, so that these same stored items can be examined by any other module within the environment (subject to a number of possible security considerations), optionally using other modules to perform conversion or interrogation duties.

By viewing the environment as being responsible and requisite for modelling and defining the application or task at hand, a number of potential extensions to current work techniques become possible. These extensions will require changes in current thinking so that users do not remain restricted. By this, we mean that users must be educated to think in terms of the Information Environment as a whole, and not be limited in their conception of what can or cannot be done, simply because - say - a given I/O device appears not to support the desired facility.

In the proposed development of a second-generation computer-based information system as the first stage in developing an Information Environment system, we are trying to break away from some established ideas such as the imposition of purely algebraic descriptions of data and environment, because it is felt that it is impossible to describe the dynamic attributes of a complete Information Environment using algebra alone.

As an example of limitation by focused perception, consider the situation of users at work with a computer system. When working at a terminal attached to a remote computer, the use of a dumb terminal operating invisibly as it provides the ability to work interactively means that the user may fall into the habit of thinking that the terminal is actually part of the computer.

In other words, the computer and the terminal are merged into one processing resource sitting in front of the user, even if the machine itself is known to be elsewhere. Similarly, some people using a small or personal computer tend to associate the 'brains' of the machine with the VDU that sits on top, even though the 'real' brain - the system unit - is quite clearly visible underneath[51]. This 'anthropomorphising' of the technology means that some users may misinterpret the relative worth of the different boxes, both in terms of monetary value as well as resources offered.

Following from this recognition of perceptual illusions, we can extend the example. Since most interaction with a computer is via a VDU and keyboard/mouse combination, there is perhaps a tendency to think of any applications also being limited

51: Cartoons which anthropomorphise computers will typically depict eyes/nose/mouth features on the screen of a computer system, reinforcing the idea of the monitor as the 'head' - and implicitly the brain - of the equipment.

to only these devices as the primary means of Input and Output, simply because these are the only interaction mechanisms. This instinctive reaction is further enhanced because the devices are designed to interact through one (or more) of the five human senses, and input to humans from any one of these senses (but particularly visual) has an extremely high priority for the human brain.

Imposing the hardware limitations upon data processing systems also promotes the traditional form of document which is text- or diagram-based, and considered to be one large entity. However, there is no real reason why this should be the case, and for the construction of an enhanced computer-based information system, we propose an alternative architecture extended to provide increased flexibility. This promotes freer thought by encouraging interaction with a conceptual 'Information Document', which describes and models the current Information Environment.

An example of this concerns group projects. Any work involving more than one person becomes a team effort, and for the best results to be achieved, the team must work effectively. A team leader has the task of overseeing the progress made, and amalgamating all results obtained, as well as ensuring the compatibility of all of the results. All of these responsibilities would be aided by the fast and effective sharing of data *and* its context, as offered by the model of the Information Environment. Each co-worker could be viewed as a contributory component of the model, building and specifying the application on a real-time basis. The use of the Information Environment means that the application can be examined quickly and easily by any authorised user at any time, and furthermore can be updated or enhanced automatically to reflect real-time alterations in circumstances as well as data.

The facility for sharing data becomes an augmented variation on the theme of electronic mail or a conferencing facility, and would permit discussions between one or more users on various aspects of the application design. But, equally, extending the same architecture by the inclusion of additional processing components (modules) would allow experiments to be carried out using duplicate sets of data, possibly with the assistance of expert system modules to simulate the working of the entire application.

The generation of the Information Document which forms a representation of an Information Environment is consequently a complex and vital objective towards the construction of a second-generation information system. Within such an environment, all possible components of the given application - users, devices and data - must be specified and portrayed. The distinction between specification and portrayal is that the specification is typeless in keeping with the ideally typeless nature of the data being processed; while the portrayal of a component within the environment is context-sensitive, depending on the nature of the component and the presentation mechanism. In effect, the portrayal of a component is an instance of an interpretation being applied to a specification.

The environment is then used to enhance the data components by introducing the representation of connections or relationships that exist between them. The value of relationships can be depicted in general terms by a simple example. Stacks of paper have indeterminate value. But, quite possibly, the topmost sheet of any stack is likely to be more important than the lower sheets. This could be because it details the nature or content of the remaining sheets. Thus, regardless of the data content, we have an identified an attribute derived solely in terms of the relationship between data objects, and so dependent upon the user's interpretation of that relationship. The validity of this can be seen on most office desks. The often seen 'clutter' of paperwork rarely has meaning or sense to an external observer, but to the originator of the mess, it has pattern and so almost any one item can be located instantly - precisely because *relationships* exist as perceived by that user.

Thus components can have relationships or patterns which may be meaningful only to those who understand or created them. The perception of the relationships can therefore portray (convey) value that is potentially as important but more abstract than the concrete data itself.

Finally, we can use the representation of the application environment to form an abstract model of the application itself. Considering each of the components as a whole means that we have effectively defined the application that is being processed. As components are added or removed - regardless of whether they are data, devices or users - so the application is altered. Sometimes, the alteration would be terminating, in that the application is halted in its current form. For example, if the user finishes a word-processing task, then the word-processor can be removed from the workspace. Nothing has ceased to exist, it is merely removed from the application, and accordingly the application has been terminated, because the environment has changed. This is a familiar concept to multi-tasking W.I.M.P. environment users, who will habitually change 'context' frequently and easily.

Alternatively, the alteration to an application may be transformative rather than abortive, as other components are added (possibly in place of existing components). For example, the user may complete a given word-processing session, and move to work with a different manuscript. In this instance, the application continues but in a completely different context.

The provision of a mechanism for modelling an application by the use of the Information Environment implies that the environment itself can also be modelled. This apparently recursive statement has a number of useful implications for information processing, specifically with regard to enhancing computer-based information system architectures. For example, modelling the behaviour of a computer network can be a complex task for simulation. But the ideal circumstances for any simulation experiments would actually be to use the real-world instance. However, any program which monitors the status of communicating hosts (as opposed to merely monitoring the passage of messages along the communications medium) must inevitably influence the behaviour of those hosts, thus altering the measurements taken[52]. Such intrusions must be taken into account, but the difficulty is in establishing how it should be done. By actually including the communicating hosts and monitor program into the environment, a more representative picture can be derived. The intrusion of the monitoring program cannot be avoided, but it can be more accurately incorporated into the performance figures.

52: This is a variation of the classic problem of whether the light always go off inside a refrigerator when the door is closed. When you open the door to look, it is always on. If you could climb inside (with suitable warm clothing), perhaps the refrigerator is 'intelligent' enough to recognise that you are inside and so the light must be left on - in effect you would be changing the circumstances of the measurement.

Section 6.3: The Information Document.

Although applications within the Information Environment will be perfectly able to produce high-quality documents in the usual sense, the second-generation information system architecture is conducive towards the production of more advanced 'Information Documents' which have enhancements that considerably improve the presentation and context of multimedia data. By including environmental data as part of the document specification, extensions or enhancements in the form of prominent or subtle effects and emphasis can be readily incorporated. The actual circumstances of the data existence becomes contributory to the value of the data, and as such must be available for inclusion whenever the data is being used.

Thus, the entire Information Environment must be modelled by building collections of data, environmental and processing components into computer-based Information Documents, which are called IDEA's (Information Document Electronic Abstraction). The IDEAs are complete descriptions of all components being modelled within the application system. This includes the user, devices, data and the working environment.

This use of global data forms in creating each IDEA means that the IDEA can be thought of as an information 'picture'. The topological and geometric relationships between the data items themselves provide considerable enhancement of the value of the material being worked with, and it is this amplification of the raw data that will improve the quality of work that can done using an Information Environment architecture.

Such IDEA's are subsets of the concepts embodied in Roszak's definition of the 'idea', which is generated by the mind in forming patterns that relate facts or data [Roszak 86].

Section 6.4: Designing an Information Document Architecture.

In order to implement the IDEA, we require an architecture for representing the information 'pictures'. Earlier work has been described in section 6.2 and leads us to envisage that the pictures could be built up using smaller and smaller data groups, which are then used to develop an extended hierarchy for the document specification. Each group of data can be collected together to form the document as a whole, in much the same way that words build up into sentences, which are assembled into paragraphs, and chapters. Attribute information would be associated with the groups for describing the environment of both the data items and the application as a whole, thus providing a more complete model of the application.

The resulting specification is clearly complex, and has the requirement for multiple processing activities, so that a multi-tasking host would be mandatory in order to support it. As a result, the architecture would also be better suited to a truly parallel machine architecture. It should not be necessary to assume that a single machine is running an application or module, and so a given application could run equally well on several machines, with application tasks allocated appropriately. An inevitable consequence of such a design is that the architecture is more suitable for the next generation of machines, where reduced costs enable greater processing power within a machine. Hence, instead of using software to simulate multi-tasking as on current technology, the second-generation information system would access multiple processors sited on multiple hosts, which would run tasks as and when needed.

All processors within the system would have two components: executable code, and local data. Their operation would be specifically hidden from all other tasks running on the host or other machines, and this would allow the processors to be developed and extended independently of each other. Although the isolated tasking concept is not innovative, the requirement that all tasks operate in terms of typeless data groups is new, and it is this requirement that provides the flexibility in the proposed architecture. Subsets of environments would be easily constructed by omitting those tasks not required for a given application.

In order to support the potential for typeless data processing, the active tasks are defined as independent execution routines, that need not necessarily carry out their task in a context dependent fashion. With few exceptions, the tasks would have to operate only upon the data supplied to them from the recognised grouping mechanism. The results of their work would, similarly expressed, be submitted as groups, either by modifying the originally supplied group; or by creating a new group for the purpose.

Specifically, as long as the task is performed in the fashion required of it, there is no obligation for it to function in a specific way. Consequently, it would be perfectly acceptable for the tasks to be developed in any desired computer language, depending only on the builders preference and experience.

In some cases, previously written tools may be preferred, and in this case special supporting tasks would perform the jobs necessary to make both internal tasks and the external tool think that their operating environments were as normal. An extension to this means that a dedicated conversion task might take the data provided in a group, convert it for use by an external tool, and subsequently convert the tool output back into the recognised format of groups. The operation of this would lie within the scope of the enhancement objective, because it would inherently support data exchange regardless of the generating or processing tasks.

Several tasks identified for use with a system may be grouped together because they have a common type of application, or for physical reasons such as being located on a specific machine within the system. For example, device processors that handle keyboard I/O, light-pen and mouse input or VDU output, could all be grouped together as user-interaction tasks. Such related tasks are also said to be assembled into a Group, because there is no requirement to distinguish data from tasks except by circumstance. Such groups could be specifically sited on a machine which is optimised for the corresponding jobs - for example a front-end processor.

Given that there could be a number of different components associated with a given application, but that there is also a desire for a common architecture to be suitable for the varied tasks, it would be preferable to have the possibility of automatic reconfiguration within the system. This reconfiguration would in the most simple situations refer to the incorporation of appropriate devices according to what was available and/or required for the application. For example, any devices which are suitable for data presentation must be incorporated into the architecture, not only so that data can actually be presented, but so that the correct (or optimal) presentation device is selected. Sound data cannot possibly be presented to the user through a visual display, but visual data could be presented in a number of ways, depending on its type. Text data presentation can appear on almost any visual device, whereas graphic (static or dynamic) data is considerably more restricted in the choice of devices that may be used for its depiction.

More complex situations or applications would require more extensive reconfigurations. This would necessitate replacing application modules or task processors with more suitable or relevant tasks, in order to perform the new application or to support alternative users.

Section 6.5: Summary.

The work by both Randell and Engelbart referred to in chapter 3 is important in justifying the extension of computer-based information system architectures to develop the higher-level abstract concept of the entire Information Environment. The technology for simple focused system development is well established, but is limited in its potential. The enhancements can only come with a re-evaluation of exactly what such systems attempt to model, and we are now finding that the models currently used may be handicapped through architectural limitations. The reappraised architecture takes into account recent research into information representation, as well as incorporating those lessons learnt from the acceptance or rejection of specific information processing ideas. Unless the ideas currently employed as the basis of systems are demonstrably useful, their validity must be questioned.

An example can be seen concerning the dominance of the IBM PC architecture within the personal computer marketplace, which is significant when compared with the smaller market share of the Apple Macintosh computers. And yet the operating environment of the Macintosh is generally accepted as being a better one than the standard PC environment - for certain users or types of application. Accordingly, the environment - and implicitly the information processing architecture - is being cloned onto the underlying hardware of the IBM PC, because it is this which is perceived as the benefit, and not the Macintosh computer itself. Thus the success of the Macintosh is perhaps not due to the hardware, but in the organisation of the processing components in providing a 'better' working environment.

Accordingly, it is suggested that a reasonable requirement for enhancing future computer-based information systems is a consistent and cooperative architecture to support interaction and collaboration between all the contributing components, which are present at many levels of abstraction from the lowest operating systems functions to the higher user-constructed directives.

Section 6.6: Research Method.

In section 6.5, it was implied that for ideas to be accepted and utilised, it must be possible to demonstrate their usefulness. This has a direct implication on the method chosen for carrying out the research described in this thesis. A formalised approach that would focus upon development of rigorous statements and definitions was considered, but it was discarded for two main reasons.

Firstly, it was felt that for a computer-based information system to be widely accepted, complex definitions and abstract arguments might deter the ordinary user. Employing formal statements and definitions during design and implementation might require their use throughout subsequent development or configuration of the system. Since a desirable attribute of an enhanced computer-based information system is that it should be understandable and configurable for all users, it would not be beneficial to require users to have a formal understanding of computer science.

Secondly, the development of formal definitions at any stage of the research would naturally influence the subsequent work, or at the very least require a significant amount of review and redesign as new ideas or concepts were established. However, the very nature of the research was such that the concepts have evolved continuously, requiring frequent and sometimes far-reaching re-evaluation of accepted conclusions.

Accordingly, it was felt that a prototyping approach to the research would be more in keeping with the nature of the work, experimenting with ideas by implementing and evaluating the results. Since a significant aspect of the work has been to develop a prototype system, then the use of prototyping is consistent with the research method. In particular, the prototype was at all times considered to be a 'deliverable'. As various components or concepts were established, they were evaluated by the author and also a 'captive audience' of postgraduate students and colleagues. In retrospect, this decision had the advantage of enabling rapid incorporation of new ideas, but the disadvantage that some of the new ideas might not always be based on sound foundations. Nevertheless, the construction of the prototype system described in chapters 8 and 9 has enabled the author to study the requirements much more clearly. The working system can be evaluated and used as a test-bed for future development.

Chapter 7.

The Information Environment and GENIE-M: Description, Generation and Control.

Section 7.1: Introduction.

This chapter considers the description and control of an Information Environment, and proposes a mechanism for its implementation. The ideal generator of an Information Environment will allow the construction of environments for a variety of applications. The environment would then allow the mixing, integration and interaction with static media types such as text and graphics, as well as dynamic types such as sound, video, film, text and graphic animation. There would also be provision for real-world control and feedback through a screen or digital and analogue switches. The proposed model described in section 8.2 uses a homogeneous geometrical data structure to represent associated data spatially and discretely. This is the theoretical basis for the construction of the prototype system. The system architecture that is proposed encourages the user to organise the sub-application environment in a more responsive fashion.

The Environment brings together the tasks of accepting data into the system, manipulating and performing computation upon that data, and presenting the results obtained. The jobs involved in accepting, manipulating or presenting data range from simple to complex, and it follows that each job being performed within the environment must ultimately be considered on an individual basis. For example, the system should be capable of handling interactive document editing and formatting operations, and providing hardcopy on any printer, and this can be facilitated using modules and a uniform data structure. The description of the environment tasks therefore consists of detailing all jobs or modules. These modules operate by manipulating data using software tools and object-orientated techniques, and also by interacting with other modules in the environment.

The flow of data and results between modules takes place within the Environment using 'Module Links'. These links should ideally connect across intra- and inter-machine boundaries, although links between machines would usually be handled by modules dedicated to supporting communication standards between machines. Module operation consists of the following three simple stages:

1) Data is presented to the module, with instructions on exactly what task is required.

2) The module performs the task as efficiently as possible.

3) The results are made available in accordance with the instructions or design of the module links.

This description means that environment modules can behave in a similar fashion to the process nodes used in data-flow diagram theory [Yourdon 78] and Petri Nets [Peterson 77]. In general, modules may only execute when all data and instructions for manipulating the data and processing the results have been obtained. Some modules will be purely computational; others take results and prepare them for presentation, or else accept data into the environment. The proposed modular architecture, which is derived on principles described in object-orientated programming methodologies such as those detailed in [Goldberg 83], [Cox 83] and [Cox 84], supports high-level integration of applications, and allows the task activities to be performed within a consistent user environment.

The presentation or acceptance of data takes place at the Interaction Boundary which conceptually exists between the user and the rest of the environment. The interaction boundary is delimited by a number of Interface Modules, which consist of software on the system side of the boundary, and (typically) a hardware link on the other side to a data source/sink such as a database or other machines. A human observer can therefore be considered to be an interactive data source/sink, thus conforming with the ideal of being viewed as part of the Environment.

As well as being a passive receiver of data into the environment, the system can be an active controller by presenting data to the real world across the interaction boundary. If the data passed through an interface module is considered to be the digital equivalent of analogue values, then the interface module can become a controller of a device in the real world, such as a lighting circuit or projection system. Such control of the real world helps in the presentation of data, and allows the system user greater flexibility in many applications.

The system has been given the name GENIE-M as an acronym obtained from the title: 'Generator of Multimedia Information Environments'. As this full name suggests, GENIE-M provides an experimental architecture for supporting an Information Environment that handles many varied types of data (multimedia data), for the purposes of dynamic presentation and interaction by one or more users. A complete GENIE-M system is designed to work with data such as text, sound, graphics, video, and so on; but it would also handle 'environmental' data: projection devices, external sensing and measurement, lighting, temperature control, and other peripherals.

As a result of collecting these various forms together, GENIE-M systems could depict and manipulate Information Environments by building descriptions in the form of the IDEAs described in section 6.3. Instead of promoting complex but limited task of data processing, the GENIE-M concept promotes the broader area of information handling within the Information Environment.

The GENIE-M architecture is ultimately intended to be a generator of working Information Environments. This is because all contributing aspects of the Information Environment under consideration can, and indeed should, be modelled within GENIE-M. Having provided the tools to model and document the Information Environment, GENIE-M could then be used to monitor and manipulate any or all aspects of an application environment through the control of the corresponding application IDEA.

Section 7.2: Geometrical Data Structures.

Section 7.2.1: Philosophy.

The prototype version of GENIE-M is an experimental data modelling system, and the key feature to the operation of this concept is the Tile. These tiles will hold the data which can subsequently be manipulated and displayed. It should be noted that use of the term 'displayed' does not imply a restriction to portrayal in visual terms only. Indeed, by using the word 'presented', we mean that there is movement of some object from a source to a destination, where the destination may be either terminal - for example a graphics display device - or transactional - for example another tile manipulation process. GENIE-M data can be presented visually to the user through user-controlled windows on the display device. Any given tile can include sound data (as well as other data types), and this sound - for example, speech - could be represented in an alternative form if that is required by the presentation device. The implicit topological and geometrical structures of a GENIE-M document is therefore apparent, rather than deliberately obscured as in existing systems. This permits more flexible manipulation of logical document entities, in terms which reflect user preferences, because the user is allowed to be more aware of the architecture. Alternative presentation schemes would make it possible to hide the underlying document architecture, if this is preferred.

The tiles are partly 'intelligent' objects, which interact with their neighbouring tiles depending on their relationship with those tiles. Each of the tiles will behave according to the relationship it is intended to portray. A tile can examine its relation with other tiles to establish areas of overlap and intersection. If a number of tiles are geometrically positioned relatively to a 'reference' tile, and this reference tile is altered or moved in some fashion, then the relative tiles must be able to adjust themselves appropriately. If one tile is moved to a new location within the tile space, then all the others must move as well. The tiles can be used for document production by more than one author. There could be many people working on the material, and such interaction leads naturally to the use of GENIE-M in multimedia conferencing. The display device windows will usually be controlled by the user, but could also be coordinated by the conference manager to ensure that the appropriate material is being presented and updated correctly.

Section 7.2.2: Physical Description.

All modules in GENIE-M interact with a homogeneous geometrical data structure, which is composed of the tiles. Within the GENIE-M system, tiles are subdivided into two forms for processing: topological and geometrical tiles. The former are tiles that are primarily used for storage of data, and descriptions of the relationships between tiles. Geometrical tiles, by contrast, hold identical data but which has been processed or formatted to take into account the geometry of the presentation mechanisms. In other words, the topological tiles are the internal record of the data and environment being processed, while the geometrical tiles are the presentation form of the data and environment. In effect, the geometrical tiles are those which are enhanced by the use of geometry in their provision, processing and presentation.

A tile can be envisaged as being rectangular in shape, and of a size that can vary relatively or absolutely with other tiles. The tiles are positioned in a stack of two-dimensional planes, called the tile space. Each tile has a tile record which contains information about that tile, such as type of contents (text, graphics, spreadsheets, sound, programs, etc), attributes (character fonts, colour schemes, language used, etc), and most importantly, the relationships with other tiles.

Any kind of data can exist within a tile. Examples of simpler data types would include text stored as ASCII (or EBCDIC) codes, or graphic images stored as bit-map or vector codes, or digitised sound. More complex tiles could hold references to devices, so that - for example - sending data to a printer device tile would effectively cause output of this data to the printer device. Similarly, a mouse device tile could hold data describing the current state of the mouse input device, while other tiles could reference large data bases facilities. GENIE-M data would be formatted prior to being placed in a display tile ready for presentation to a display device, with the kind of formatting used dependent on the type of display device. As an example, a tile could hold digitised sound, such as speech, which cannot be meaningfully 'seen' on a printer or VDU. The speech display data could either be presented (correctly) through a sound output device, or else a speech icon would be presented to a printer or some other visual display device.

Tiles can contain three main types of object: raw data, formatted data (for display), and reference or executable program code. Depending on the primary use to which they will be put, such tiles are called Data Tiles, Display Tiles, or Reference/Program Tiles. Reference/Program tiles would not normally be displayed, display tiles would not normally be used for holding raw data, and data tiles would not normally be executed. However, tiles can always be handled by modules in the same way, regardless of their content. Thus tiles are as global and typeless as possible, not restricted to holding only data.

The data held within a tile can be dynamic. For example, a data tile could hold a reference to a current share price for use in a financial report. Each time that the tile is accessed, the latest up-to-date value would be obtained. Under suitable circumstances, any references to that dynamic tile would cause real-time changes to the presented results. A reference/program tile could access a data tile holding references to spreadsheet values, and take the data to produce, say, a pie chart which is constructed and held in a display tile. Reference/program tiles will often bind closely to one or more data tiles, although they can also be for general purpose use by the system.

The geometric positioning of display tiles in the tile space reflects their actual location within a particular document or application. The position is represented using the topological relationship between tiles described in the tile record, in a similar fashion to Knuth's concept of nested boxes connected by stretchable glue in the $\TeX$ system [Knuth 83]. Tiles cannot overlap on the same plane and are affected by any changes in size of neighbouring tiles. For example, a display tile of graphical data which is located below another display tile of text in the tile space would be placed accordingly in the document presented on the display device.

Tiles holding text data to be formatted for display must also take into account the effect of that formatting on related tiles. For example, if a tile is required to divide its text into a number of columns with a diagram in the centre, then the formatting must allow for the size and position of the diagram. If the diagram changes size, then the text layout will probably need to be altered as well. Similarly, tiles that hold sound data should be able to detect two or more voices 'speaking' at once. Formatting takes place by 'flying' through the tile space, examining all those tiles required for the presentation. A 'filtering' process may be carried out at the same time, to identify required or unnecessary items of data. As each tile is 'seen', its formatted contents are added to the display tile, ready for the end result. In practice, a number of sub-display tiles may be used to build up the final display tile, so that small (local) changes to tiles do not require such a major change to the overall presentation.

Tiles may themselves refer to sub-tiles. If a tile contains text and it is desired to emphasise certain portions of the text (possibly by *italics* or underlining), then either the text could be subdivided into more and smaller tiles, or the tile could have sub-tile references which hold the emphasised portions of text. Tiles which store animation material will bind very closely to each other. Although each is an independent unit, any alteration made to one of the tiles in the group will probably affect other tiles within the same group. The method by which animation is achieved can vary - the simplest technique would be to have a number of 'frames' stored, one per tile, and the frames can then be cycled, in a similar fashion to that used in ordinary cinema films. The frames would be stored in a 'stacked' fashion within the tile space, so that by 'descending' through the levels of tiles, each frame can be displayed in turn.

The organisation of tiles, and the relationships that are explicitly or implicitly formed between them, are all referred to by the generic name of a 'Web'. This concept has certain similarities with Knuth's cweb model [Knuth 84], in that it implies the existence of a structure relating and controlling all the entities within the formation. However, the GENIE-M concept of the web is a much less formal concept, more useful for envisaging the geometric modelling of tiles within the tile space.

Section 7.3: GENIE-M Architecture.

Section 7.3.1: Modules and the Environment.

All modules within a GENIE-M environment have the same structure: executable code, and local data. Their operation is specifically hidden from all other modules or tasks running on that or other machines, allowing modules to be written independently of each other. This means that subsets of GENIE-M environments are easily constructed by omitting those modules not required for a given application. A further advantage of the modular architecture is that GENIE-M is well suited to making use of multi-processor computers, as opposed to running on a time-shared basis on single-processor machines.

One unique module which must be present is the central administrator, or 'hub'. This module has the same structure as the other modules, but rather than manipulating data, supervises the relationships between modules. The hub does not oversee modules, because each module normally functions independently of any other module. However, the hub does examine the overall running of the system to ensure that no problems or deadlocks occur. The hub also has the responsibility of introducing and removing modules from the environment, and generally serving the needs of all modules present in the system. The hub maintains a record of which modules are present in the environment by using its own geometrical data structure. Any modifications to the environment are reflected by updating this structure[53].

Several modules present on a system may perform similar or related tasks. They may be grouped together because they have a common type of task; or for physical reasons such as being distributed between one or more other machines in the system. For example, modules that handle keyboard I/O, light-pen and mouse input or VDU output, could all be grouped together as user-interaction modules. Such related modules are said to be grouped together into a Sub-Environment. Just as the main GENIE-M system has a hub to control the environment, so each sub-environment will

53: In this way, it is possible for the information environment to 'examine' itself, by looking at the stored model of the environment.

have its own Sub-Hub. Since the hub is responsible for activating all modules in its environment, a bootstrapping technique can be used in initial construction of a GENIE-M system, and later activations.

Section 7.3.2: Tile Transfer and Manipulation.

Each module within the environment only inputs or outputs data in tiles, so that a module can work with any kind of data, regardless of what it actually represents. The exceptions to this are the interface modules which accept raw data for conversion into a tile format, or present results having extracted them from display tiles.

A tile which contains - for example - bit-map information could readily be transferred to another module which manipulates the bit-map, without actually knowing what the data is. In a simple case, the bit-map image would perhaps be 'reversed'; in a more complicated case, the image could be 'skewed' or 'slanted'. Tiles can be read and converted by Intermediary Modules which are used to supply data to programs or machine processes that would not otherwise understand data tiles.

Section 7.3.3: Module Organisation within the Environment.

The main feature of GENIE-M module organisation is the hierarchy of module invocation. Initial work in preparing a GENIE-M prototype has suggested that a typical system will have a large number of fairly small modules performing elementary tasks. A system which retains modules on a most-recently used basis will allow frequently used modules to 'float' to the top of the priority table, giving faster overall response. Most modules will connect to only a few other modules, and typically two connections will provide input, while one or two routes produce output.

The organisation of modules is therefore very important in order that data may flow as quickly and as efficiently as possible around the environment. The data communications concept of a 'packet' as the primary data and action structuring mechanism is reflected in a similar message passing technique used by modules for working with the GENIE-M geometrical data structure. At the 'centre' of the environment will be the hub module, and around it is (conceptually) placed inner service or core modules. These core modules will tend to perform tasks of an administrative nature within the environment. Frequently, they will initiate activity without user interaction. Most sub-environments will be located near to the hub, since they form a natural division of tasks into related areas - they also follow a similar organisation to that being considered. Further from the hub, elementary rather than compound modules or sub-environments will begin to predominate. These perform tasks of a more operational nature.

Finally, nearer the interaction boundary, the modules will become more machine dependent in nature with hardware connections and dependencies. The majority of user interface modules will be located here, because they are closely linked with the interaction boundary.

Section 7.4: The User Interface.

The users of GENIE-M will need sophisticated software tools for entering and viewing information. This support is likely to include editors for knowledge networks, or composition tools for creating useful video or audio sequences and animation. The GENIE-M infrastructure allows convenient communication with editors and users at remote sites.

The 'User' is a general term used here for both the human user and the display device being utilised to portray GENIE-M data from a display tile. The display device is defined to consist of objects such as vocoders for sound reproduction; VDUs for visual information; keyboards, mice and light-pens for information and command entry, and so on. The GENIE-M user interface adopts the dynamic representation features which are facilitated by the tile architecture (as discussed in section 7.2) for circulating data within GENIE-M system.

At the user interface level, all internal communication tasks are performed in a manner transparent to the user or users by the interface modules of a particular application. Each of these modules within the system has a specific and clearly defined set of tasks to perform in order that GENIE-M data may be edited and presented as desired. An example is the window administrator module, which is one of the fundamental components in GENIE-M. This module has a full knowledge of the devices under its control. For every display window or viewport, the window administrator will keep an internal record of its structure, current contents, inter-relationship with other windows, ownership, and other details. This module is primarily responsible for presenting display tiles on the display device by creating windows and removing them when not required. It therefore 'knows' about the geometric and implicit topological organisation of data for a specific application, in order to map accurately the data onto one or more display windows.

Another module of importance is the geometric structure editor, which manipulates, creates and organises tiles of data in a tile space. It is capable of grouping tiles into larger structures such as a tile plane. Inter-relationships between tiles, for example, cross-referencing of data within an application, or even among applications, are defined in this way. This type of linking, cross-referencing and ordering of data in some instances resembles that of structure editors [Kimura 86] and hierarchy editors such as PEN [Allen 81]. Accordingly, the GENIE-M geometric editor has features that are in common with the hierarchy editor concept. Some editors include facilities for cross-referencing to show inter-structural linking, though these cannot be explicitly shown in the hierarchical structure.

However, GENIE-M's linking capability is implicitly portrayed by the geometric structure. Using a geometric representation for the relationships of tiles within the tile space allows us to depict both cross-linkage of data and relative - and hence topological - data organisation. Both of the cross-linking and topological concepts used in GENIE-M are the natural and preferred products of one uniform structure, and reflect similar concepts found in the ZOG [Akscyn 84 & McCracken 84] and Textnet [Trigg 86] systems.

The geometric structure, when regarded as an abstract data structure, has an additional advantage in providing an implicit indexing scheme. An indexing scheme is a technique of organising data for the purpose of accessing it efficiently, and can be used to assist a human user in the search for certain information. Tile attributes are inserted using a generic tile template that is accessible via menus available at that stage of the interaction. Specific tile templates may be maintained for particular applications.

The human user forms the highest level of GENIE-M administration and control. Tasks performed within GENIE-M are initiated by real-time events. A major subset of the real-time events is the issuing of commands by one or more users while directing the tasks performed by the system. Alternative real-time events would include the continually changing time-of-day, or the current status of peripheral devices. Human users 'interact' with an application via GENIE-M windows, which in turn associate with relevant display tiles. The means of user interaction are usually locator devices - a mouse is the most likely choice or a keyboard on a windowing system. Other means such as a vocoder could be easily introduced into the proposed system. The windows are used to depict the current state of the system in a manner determined by the specific application. A monitoring task would depict the multiple processes or activities being monitored, while information retrieval tasks would depict the requested data (assuming it could be located).

Section 7.5: Application Examples.

Given the complexity of the concept, it is a good idea to give some examples of how the system works. The process of understanding the GENIE-M concept is helped by considering diverse examples of uses to which it can be put. We therefore elaborate upon the GENIE-M concepts in order to clarify any confusion which may occur while trying to understand GENIE-M. Following this are some detailed views of differing applications where GENIE-M would offer a powerful range of abilities in coping with the given task, as well as providing ample potential for further expansion in that area.

Section 7.5.1: Mixed Media Documents.

GENIE-M's philosophy is realised by applying interactive multimedia documents for presentation, manipulation, storage, and final hard-copy. The geometric data structure interface allows the human participant to interact with presented data. The participant selects relevant data by traversing the geometric planes using a locator device (a mouse, foot-pedal or cursor keys, etc.). The inter-relationships between different data objects (for example, text or dynamic graphics) within a document are represented by tiles in a geometric space. The tiles are associated with processes such as text or graphics editors. At any time, the participant is presented with an accurate portrayal of the portion of data associated with selected tiles. Data hiding could be effected using access rights defined on the tile for data security, or by simply 'hiding' the tile 'behind' other tiles - an action that it facilitated through the use of the geometrical organisation.

Modification of or addition to the currently presented document is performed by editing old or inserting new tiles within the geometric space using a geometric editor at the interaction boundary. The tiles are associated with processes created and coordinated by the hub. The geometric planes provide a simple mechanism for generating and organising animated graphics, text or images in a dynamic document structure or lecture scripts.

Section 7.5.2: Newspaper Layout Construction.

Daily newspapers have many features in common. The majority of the information and news is printed in black ink on white paper, and so in order to present the reader with some variety and also to catch the eye, a number of type-sizes, fonts, styles, and photographs are included. Each of the main news stories has its own block somewhere on the page, sometimes with a relevant photograph placed at some location within the block. It is not always the case that such a photo would be in line with the block of text, and indeed photographs to one side of the text (and so forming an 'L' shape block) are common.

Given the somewhat haphazard nature of block layout, it is a difficult job to lay out the page in a readable and presentable fashion, while fitting in the maximum amount of information on the page. Furthermore, as news stories are updated or invalidated, the block could be altered by being increased, or reduced, or removed altogether.

There has been considerable discussion recently [Honeywell 87] over the use of high-technology in producing newspapers, because such use would enable the layout editor to perform the task more effectively and quickly. However, the techniques being introduced make little attempt to integrate the entire process - if a block is being manipulated on a VDU, it is unlikely that the contents of the block will be considered, and if a block is later re-written then the layout process must be repeated again to ensure that the material still fits.

GENIE-M would allow the concept of newspaper to be broken down into smaller and smaller logical units: newspaper to pages to blocks to paragraphs to words, and so on. Each one of these units corresponds to a GENIE-M tile. A high-level tile (for example, the page tile) would hold references to lower tiles (the blocks on that page). Block tiles would refer to paragraph tiles holding text, or possibly photographs. Each one of these tiles would be individually edited by the journalist, directly into the GENIE-M newspaper database. As the tiles are entered into the system, the section or feature editors can combine the tiles into tile groups, for example all international news items would be collected together into a local number of pages.

These tile groups can themselves be built up into groups to build the entire paper. Each of the tiles is at this stage a topological tile - the layout is comparatively free, although there will naturally have been specification of paragraphs, emphasised text or fonts, position of photos relative to the text (for example, between two text paragraphs, or to one side of a paragraph or another photo). Most importantly, the entire news item is considered to be one unit, and the author is rarely required to consider the possibility of splitting the text over a number of pages. If editorial constraints require that the material must be split, then a tile would be separated down into two - or more - subtiles.

Once the topological tiles have been entered, the GENIE-M system can begin automatically formatting the document. It will know from the structure of the tile web which tiles relate to each other, and so it can build up the geometric tiles knowing the contents and attributes of a tile, and also its relation to other tiles. In the event of a tile being too large for the available space, the GENIE-M system can create a new tile to hold the extra material, and this new tile can then be submitted into the tile web for another convenient page, along with bi-directional cross-referencing of the form 'continued on page XX' and 'continued from page YY' (possibly including column numbers also).

The author of each item will have already specified information such as type of tile contents, paragraphs, size of font, type-face, and other details. The section or feature editor will have included further information regarding the relation of items to each other - for example major news items would be expected to appear above and/or to the left of minor items. Within these constraints, the GENIE-M system will construct geometric tiles that reflect the individual layout of blocks, as well as the combined page layout of all the blocks on that page. The display of these tiles allows the overall view of the page to be considered, as well as the layout within each of the blocks. If any one block needs changing, only the relevant tile needs to be altered, and then the reformatting process to re-construct the relevant geometric tiles can be repeated.

Time-sensitive material can easily be included - for example many newspapers carry information on stock prices or exchange rates, and topological tiles can be entered into the document that describe the layout of the information they carry, as well as references as to the source of the information, but not the actual information itself. At any time when any geometric tile is constructed which uses such topological tiles, the cross-reference is followed and the latest and most up-to-date value is automatically inserted.

GENIE-M is ideally suited to the high-speed and quickly changing environment of newspaper publishing. Since all contributors to a paper (in whatever context - journalist, editor, and others) all have direct access to the relevant portions of the paper, which can then be combined and manipulated within the same system, the overall effect is of a much higher production rate, at lower costs to both the publisher and the reader. Since GENIE-M is context-sensitive to the display device being used, the material can be formatted appropriately for the device - for example if a publisher wishes to place selected material into micro-film format, the exact same geometric tile construction process is followed, but using the micro-film device filter. Each one of the topological tiles would be stored in the core and later archived into a news file, and the very nature of the topological tiles makes cross-referencing an easy and trivial task (albeit on a large scale).

Section 7.5.3: Lecture and Presentation Control and Structure.

Better lectures and seminars have a structure to them. After an initial introduction to the topic being discussed, the main body of the presentation consists of a submission of fact, information and views, which may possibly be discussed among the group. Finally, a summary and conclusion is presented. Throughout, there is often use of diagrams and pictures to help explain ideas or portray facts more clearly. The diagrams could be in the form of slides (and overhead projections), films, and (more recently) television and video. Some presentations may also draw on audio material from tapes or records.

Given that the material and portrayal of material has a structure, it becomes apparent that GENIE-M offers a number of useful capabilities in this application. The entire lecture could be prepared by the lecturer, with each constituent object being allocated a topological tile.

For example, each section of text in the notes could be allocated its own tile. The tiles would be displayed to the lecturer in turn (using some form of head-up-display), who can then present the material - alternatively, if the lecturer is running behind schedule, it would be an easy matter to skip onto the next section, knowing that all the cross-referencing of related material will be carried out appropriately.

Other tiles could be used as 'channels' for the input of information from outside sources. For example, in a large auditorium, televisions may be used to display camera views of the lecturer or diagrams. A camera could be used to 'channel' the image directly into a GENIE-M tile. The lecturer then has real-time control of this image around the auditorium, and as each new diagram is to be displayed, can simply direct the appropriate tile to the televisions. A tile which holds pictorial information for projection on a large cinema-type screen could have an associated attribute file that would automatically dim the hall lights while the tile is active; once finished with the image, the lights could be restored to their normal intensity. Alternatively, the lecturer could have a 'control panel' presented on the display device so that personal control of the environment can be obtained as and when desired.

The display device for the lecturer's use would ideally be of the 'electronic lectern' device frequently seen today at major conferences and seminars. This device has a transparent display screen in front of the speaker, onto which is projected the text of the speech from a projector underneath. The angles involved in this 'head-up-display' mean that the audience does not see any of the speech notes, and further the transparency of the screen means that the speaker is not obscured in any way. A small foot control device could be installed to control the speed of the presentation of material onto the screen, thus leaving the lecturer's arms free.

Setting up such a lecturing system would require that the lecturer arrive early with a technician to connect-up the appropriate apparatus and its control mechanisms. As each device is identified and a driver incorporated into the GENIE-M system, the driver can be permanently kept on record for later re-use.

With all the material on hand in this form, the lecturer could display actual copies of the notes in the form of projections, simply by redirecting tile contents to the display device. A mobile pointer (possibly driven by a mouse) could be superimposed on the projected image, in a similar fashion to a multimedia conferencing system. Finally, having all the material stored in this way would allow hard-copy notes for the audience to be produced easily and quickly.

The construction of the lecture notes would be made easier by the probable existence of a large amount of material already in GENIE-M tile format - it would then be a simple matter of constructing a new lecture web from the information tiles already stored in the core.

Section 7.5.4: Prototyping Methodologies.

When designing any system or device, it is useful to be able to build and study a test system or prototype. The prototyping stage allows a more detailed evaluation to be made of the system at an earlier stage of development, prior to a more detailed (and probably more expensive) implementation of 'the real thing'. In the case of data processing tasks, where information is continually being moved and manipulated in various ways, it is important that the movement and manipulation is carried out in a clear and correct fashion. For small models, this may be easy to describe and implement, but for larger models, the possible types, quantities, sources and destinations of information easily become impossibly large to wield and test.

GENIE-M offers a structure mechanism that is designed to cope with large inter-relating and interacting items of information. A web is constructed which shows the relationship between any number of tiles. If the tiles become items of information, and the connections are used to represent the flow of this information about the system, then GENIE-M becomes a powerful prototyping methodology for the design and implementation of such an information web. Command modules could be created that will accept this web as input, and either mimic and so portray the flow of information, or possibly produce statistical results as desired.

Section 7.6: Problems with the Concept.

For a developed version of GENIE-M we would choose to use a menu-driven, windowed user interface that makes use of the 'cluttered desktop' concept of overlapping windows (examples are Xerox Star, Apple Lisa and Macintosh, etc). Such a depiction can rapidly become confusing due to the many objects displayed.

However, the user or users currently directing the tasks will be familiar with the steps taken to reach the current state, and so in effect the multiple objects on the screen - which relate directly to objects within the tile space - can be considered to be a 'viewpoint' from the current perspective into the web.

The mechanism used by the windowing interface to associate with the underlying geometrical model must allow users to transfer information between windows in more structured manner than that provided in less developed systems. For example, a user can work on the text for a document in one window, and draw a figure for the document in another, which can then be transferred to the text window at any selected position in the document. Furthermore, each user would have personalised sub-tile-spaces, a set of common tile layers that are distinct from, but associated with, the common tile layers that all users have access to. In this way, multiple users can share tiles by placing them in common tile space, while private or individual information can be located within the local tile space.

A more obvious problem concerning the concept rather than the implementation pertains to the initial creation of the multimedia document or IDEA. The authors who have contributed to the material to be displayed clearly have an initial responsibility for its accuracy, but subsequently, as other authors contribute, or the scope of accessing the material increases, so the 'ownership' of the IDEA may be increased. The problem of updating the IDEA becomes increasingly difficult, especially with regard to ensuring integrity of the stored material.

One temporary solution might be to archive permanently the original material and all subsequent alterations. However, for important or popular IDEA storage, this would rapidly become unwieldy. In effect, the concept of the IDEA becomes akin to the 'customised' working environment, where each user has their own 'version' of the application. While the potential for multiple and conceivably complex structures is responsible for posing such problems, to reduce the same potential would in effect be to limit the flexibility of the system and the architecture. This latter option would counteract the perceived benefits of flexibility, as discussed in section 5.4.

The actual implementation of the architecture may appear to be a retrograde step, in that there are certain aspects which redesign existing systems. However, this is considered vital in order to avoid the problems inherent in 'patching' systems: all too often, attempting to patch an existing system will result in a new system that remains constrained by the limitations of the original. If the limitations are to be specifically removed, this would usually require a considerable amount of work; and would in any event lead to potential conflicts with the original system design. The less 'efficient' approach of identifying the essential aspects of the system before commencing implementation results in a design architecture which reflects the original specification much more accurately.

In terms of implementation, however, there are a number of aspects which appear to be derived from the domain of operating systems, rather than application systems. For example, certain elements of the data or tile handling, or the use of drivers for specific presentation devices. [Scott 86] has said that rather than including device specific code in an application, device drivers accessed through a virtual device interface (VDI) provide:

> 'applications with peripheral independence and system integrators with a clearer route to completed applications.'

This can be further justified by the realisation that in order to provide a more enhanced form of integration - as necessitated in any attempt to extend computer-based information systems - we require that the distinction between operating system and application system must be made less distinct in order to provide a better representation of the working environment. [Halasz 88] has pointed out that joining an application system closer to a support system is likely to produce a more efficient result, although the actual joining may be more complex than keeping the two systems distinct. Partitioning tasks between several domains means that interfaces would also have to be defined, when in fact we are attempting to provide a contiguous model of an operational environment.

The approach used to envisage the individual application modules is rather similar to the 'black box' concept of current software engineering techniques, whereby the operation of a component may be safely ignored and will often be specifically hidden. This may be contrary to existing standards such as Office Document Architecture (ODA), where an 'open architecture' is seen as essential to promoting standardisation.

However, it is often the case that such standard architectures cannot benefit from environmentally-specific advantages. Constraining a system so that it must use a given standard must at the very least reduce the overall performance of the system. By contrast, using a proprietary architecture means that the system can be optimised for specific circumstances, and when necessary conversion or environment-support modules can be applied.

Finally, the actual implementation of the experimental system has made clear the need for advanced hardware technology, particularly with regard to true multi-tasking and parallel processing. In order for the proposed system to run at optimum performance, there is a requirement for advanced host machines that are not as yet available, although the indications are that such machines may appear before long.

Section 7.7: Conclusion.

GENIE-M is an implementation of an architecture for developing computer-based information systems. In order to increase its power and capability, it has a number of language-like and Operating-System-like features, but it is still nevertheless a system for data processing and editing. GENIE-M's basic units are tiles. These (and only these) are what GENIE-M components are expected to handle or manipulate. However, GENIE-M applications are not required to give consideration to the actual information that is contained, portrayed-in or referenced by those tiles; except in those situations where interpretation is mandatory. As a result, tiles can be data of any kind, or command files, or programs (source or executable), spreadsheets, processes, and so on. This enables a more generalised approach to be used in system construction. In order to make the handling of these tiles both easier and more powerful, certain specifications have been included within GENIE-M such as allowing external execution to be invoked (for example, remote procedure calls).

GENIE-M is very much suited to a parallel architecture - it has in fact been designed that way. GENIE-M does not have to assume a single machine running the system, and so GENIE-M could equally well run on multiple machines (one dedicated to each task). For example, we envisage the probability of having one machine per display window as being as likely as one machine running 'n' display windows.

The thinking has been in terms of the true next generation of machines, whereby rather than using software to achieve multi-tasking, in fact the system would have multi-processors which could run tasks as and when needed. The problem is getting such parallel-ism working, but the GENIE-M concept is *designed* for parallel-ism.

It is realised that GENIE-M might never work efficiently or quickly on a single machine system - and it was neither intended or expected to. However, subsets of GENIE-M could run on present systems, in a similar way to having word-processors or spreadsheets performing jobs one at a time, depending on which program modules you have installed on your system.

Section 7.7.1: Use of GENIE-M on Existing Systems.

GENIE-M can be used by existing systems. A possible objection to this might be that depending on the construction of the existing system, there could be a need for a considerable amount of work to convert formats and file-types, so that GENIE-M and the existing system can communicate. However, GENIE-M is intended not only to use its own general-purpose, multimedia editor, but also to invoke any external system and supply it with the desired input, and then re-direct the output as required. GENIE-M would effectively simulate the environment in which the external application would normally expect to run.

It is possible that the size and complexity of the final system would increase, but the degree to which this is true will be environment dependent. If there is likely to be regular use of a non-GENIE-M facility - for example an external word-processor - then it would be more sensible to remove the unused portions from GENIE-M. It has been described elsewhere that GENIE-M is constructed from modules, and if any one of these modules is either not used, or already implemented as (say) an operating system command or through use of an external program, then it would be sensible to make use of the 'host-supplied' tool.

Conversely, if the intention is to move fully over to a GENIE-M system, then after an initial conversion phase for formatting information, the older system could be removed in the same way as redundant modules are removed from GENIE-M. This would not reduce the overall performance of the system, unless GENIE-M is being prevented from operating at its full potential, in which case the older application would slow down the system, rather than GENIE-M being the cause of any bottleneck. If it is desired to use video, sound, text and graphics all in the same document, then it is inevitable that formatting will be required, and further that only GENIE-M will be able to handle the resulting new document. This formatting would normally be one-way (since a text-only editor could not handle the intricacies of a multimedia document), although because GENIE-M holds each type of information in separate tiles, it would be reasonably straightforward to extract the text-only portions of a document and submit them to a text-only editor.

Section 7.7.2: GENIE-M Users.

Those users who are already using a system which performs some of the tasks that are supported by GENIE-M will naturally prefer not to have to learn new techniques and instructions. For GENIE-M to be used in a multi-user environment, it must offer similar or better performance than their existing system(s). However, GENIE-M's module concept makes not only upgrading easy, but the entire environment can be altered as the user desires. Each module can be individually tailored, so that whenever a user invokes GENIE-M, the appropriate modules are loaded into the local workstation for use as normal. These modules can be user-written, in any language, and could easily be OS or system calls if they are suitable. It must be stressed that GENIE-M is not a system, but an architecture. The distinction is that using the architecture and the generating tools allows users to specify, develop and configure their own personalised systems and applications.

Section 7.7.3: Future Systems and the GENIE-M Architecture.

It is generally recognised that with decreasing costs for more powerful hardware, the machines of the future will offer greater processing capability, possibly in the form of multiple-processors within the machine. GENIE-M is organised around such an architecture, and indeed presents a viable mechanism for organising the multiple-tasking activities of the machine. The important feature for such machine organisation is the internal communications to ensure that parallelism is achieved where possible, but that events do not clash or cause deadlocks of resources. Using the GENIE-M system as a design approach should allow a software test-bed to be quickly implemented as final hardware.

The GENIE-M model has the following objectives and advantages over non-integrated systems:

1) Providing an integrated framework for handling data from diverse media types in a coherent fashion.

2) Specifying and directing the manipulation of objects and relationships stored within a geometric data structure.

3) Aiding the user in the processing and presentation of data using simple structure traversing and editing or formatting techniques.

4) Monitoring and controlling the movement and modification of objects within an application environment, using access rights associated with parts or the whole of the object.

5) Ensuring efficient use of the processing powers of different devices during data preparation and presentation through a modular architecture.

6) Providing multiple-process message-passing primitives for intra- and inter-communication between data processing modules.

The model is widely applicable in many areas: a computer conferencing system where the ordered control of information is important; electronic multimedia documents; photo-typesetting; distributed and semi-automated lecturing sessions where control of lecture devices (e.g. lectern, slides, projectors) and lecture notes are to be pre-specified within the document structure for interactive presentation; newspaper publication requiring a structured layout of information using current computer technology.

Chapter 8.

Design of the Prototype System.

Section 8.1: Introduction - Analysis and Design of the System.

In the first chapter of this thesis, we identified the goal of this research as being the production of an architecture that will support the development and operation of a second generation computer-based information system. In order to do this, we have had to identify in general terms what is actually required from such an information system, particularly with reference to the human users of such systems. A working definition of information has been given, and used to define a computer-based information system as that which enables and supports the computerised processing of information for human users.

Having established these concepts, it became important to identify the architectural features that a generalised computer-based information system should have in order to eliminate many of the current constraints found within the first generation examples of such systems. Consideration was given to examples of current production packages, such as management systems, data handling tools, and idea processors; as well as to the cases of research work that have not necessarily seen commercial success as yet, such as multimedia systems, hypertext[54] and information structuring systems. This enabled us to isolate those features that seem essential ideals or key aspects of the tools, but which were not always incorporated as part of the end-result of each of the systems studied[55].

54: The Apple HyperCard product being a notable exception to the lack of popular acceptance for research products, although it remains to be seen whether it proves to be a resounding commercial success.

55: For example, a graphic user interface is not provided with the HyperPad systems, although this package implements most other features of a Hypertext system.

In effect, we were able to establish the fundamental features that seem mandatory for the construction of an enhanced or second-generation computer-based information system architecture. The features were categorised and grouped together appropriately, so that an information interaction diagram was produced, thus forming a pictorial statement of the architectural requirements.

With the architectural requirements defined, the next stage in the development was to identify a suitable architectural mechanism which would support those requirements. A uniform and consistent model was required, and the result was called the GENIE-M generator of computer-based information systems. This provides support for the concept of the information document, a computer-moderated instance of an information-processing application, which is modelled using electronic abstractions of the data, relationships between the data, and the environmental circumstances of data and the relationships themselves. Chapter 7 discussed the architecture in detail, showing examples of how such a structure would operate for a variety of cases. However, while the structure has been identified in its abstract form, the major task remaining concerns the development of a prototype system to evaluate this architecture. This done, experiments can be carried out to test and improve the model to the point where a second-generation information system architecture is firmly established in terms of current work.

Section 8.1.1: The Contribution and Incorporation of Software Engineering Techniques.

In section 2.3, we examined the concept of development systems, and discussed the contribution of computers to Software Engineering (C.A.S.E.). Although it was established that there are limitations to the concept, there are nevertheless many reasons for following an organised and structured development process for the construction of the prototype system. Firstly, the requirements for most systems (particularly commercial applications) change far more rapidly than the corresponding software can be developed. Secondly, the users who issue the requirements hardly ever know in advance the functional details that are required from a new system - until it becomes clear that they are missing. Accordingly, if a system is to gain or retain its value, the development methodology used for its production and subsequent maintenance has to be able to respond quickly in accommodating new directions. The application of software engineering techniques has been intrinsic in supporting the design and development process of the current project.

However, while recognising that methodologies already exist which enable the developer to establish a full specification and implementation plan from conception almost to completion of the finished product; there are alternative approaches which are not so definitive in their strategy. This is inevitable because there are several instances of systems which cannot be specified in advance[56].

In the absence of a suitable specification, it is difficult to develop a design or production strategy. However, current thinking in software engineering makes allowance for systems which cannot be specified in advance, and there are techniques for the design of such products.

Specifically, the concept of prototyping [Gomaa 83] is used for certain kinds of system development and design, and seems to be of particular relevance to the research being carried out in this project. The difficulty is that for many new systems, and particularly those for which the concepts embodied can be large or complex, it is almost impossible to make any form of meaningful assessment of validity before the system is built and put to use. Rephrasing in general terms, the problem is to carry out evaluation of a complete system, without necessarily expending a great deal of resources on implementing the complete system in the first instance. As Jones indicates [Jones 89]:

> 'Prototyping offers hope to software developers faced with the complex problem of having to provide less than prescient users with computerised support in next to no time. By enabling I.S. staff and users to "grow" applications, prototyping supports an iterative method of software development.'

56: For example, Expert Systems or those designed to offer Artificial Intelligence.

Accordingly, the scope of this research was extended to incorporate the associated issue of evaluating the benefits of a prototyping approach in general terms, as well as in the specific instance of the implementation of the prototype system. Of particular interest was the study of the handling of development in two categories:

1) Management of the system development. Each developer (system internal and graphic front-end) was totally independent, to the point of working on separate sites with only telephone links to communicate. Actual meetings were held occasionally, and were used to provide progress reports and encouragement, as well as to discuss problems in development.

2) Incorporating changes into the architecture, while recognising that a major concern is that of quality, not productivity.

Section 8.1.2: Evolutionary Prototyping.

In general terms, prototyping is the adoption of an exploratory form of system development, where the products are knowingly incomplete, but which can be modified and augmented as the real requirements or modified specifications are derived. Two major forms of prototyping can be identified, the first being to modify and improve the prototype until it reaches the point where it meets the specified requirements - which may themselves be changing during the development process. The second form of prototyping is deliberately to build a 'disposable' system, which can be jettisoned as and where necessary. In this form of prototyping, a (possibly expensive) model is built to identify problems or areas of extended benefit, preparatory to discarding the prototype completely in preference to a 'production-quality' version. As Sommerville indicates [Sommerville 89], the appeal to patch and improve the prototype in order to retain it may be strong, primarily in a desire to minimise development time and costs; but this temptation should be resisted because:

1) Some system characteristics may have been deliberately omitted, such as security, performance, robustness and reliability.

2) Changes made to the prototype reflecting user needs may not have been made in a controlled way. Thus the only design specification which exists is the prototype code - an inadequate basis for long-term maintenance.

3) Development changes may well degrade the system structure, making maintenance more and more difficult.

The mechanism of prototyping depends on the nature of the task at hand, and almost any software tool can be used for certain aspects. For example, any screen painting package could be used to design the appearance on the screen of the final product, long before it is finished. More complex tasks generally require more complex tools, however, and current tools specifically intended for prototyping are frequently built upon powerful 4GLs and code generators. One of the problems with this, however, is that while 4GL technology, in particular, is in reasonably widespread use within information centres, it is still unproven for building the core of strategic systems. Certainly, 4GL-developed applications consume more resources[57] than 3GL equivalents [Gibson 89], and further tie the developers to a proprietary (focused) system. However, if the prototyping approach is used as intended, to produce a throw-away model at the end, then the disadvantages are considerably reduced.

In the case of the present research, a considerable amount of thought and consideration has been applied to establish the architecture, but there are a number of questions that simply cannot be answered without recourse to an examination of a working example - does it work as intended, what are its practical limitations, is it adequately defined, etc. Therefore, a major goal in developing the prototype generator system is simply in order to help with assessing the validity of the proposed generator architecture, thus aiding evaluation of how the existence of ideas embodied in a production system would affect the development of future computer-based information processing systems.

57: Including processing time.

Section 8.2: The Structural Specification.

In order for the system to function efficiently and quickly, the design of the architecture must be considered carefully. All items that form part of the working whole must be defined with a view to avoid the danger of inadvertently incorporating restrictions. From the system implementation perspective, the architecture which operates within the environment can be split into three basic sections, as depicted in Figure 5.

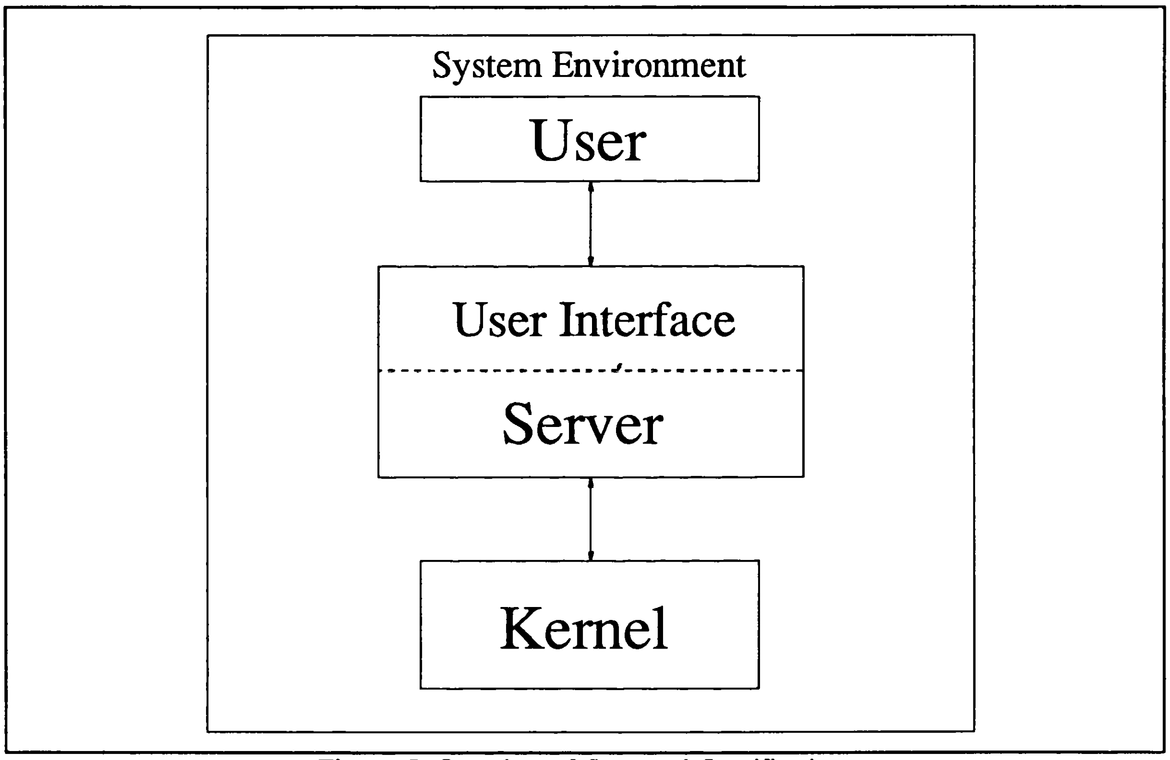


Figure 5: Overview of Structural Specification.

The User component represents the user aspect of the interaction diagram referred to in section 5.2. It delineates the domain of presentation from, and command submission into, the computer-based domain of the remaining system environment. The computer-based domain consists of the User-Interface/Server and Kernel domains. Interaction with the computer-based domain is implemented via a series of interaction tools, depicted in the diagram by the User Interface block. This section is responsible for handling all communication between the internal GENIE-M system and the user. Closely associated with this block - and in many ways indistinguishable from it - is the server domain. This section contains the specific application tools and modules which are necessary or desirable for the tasks at hand. Higher level administration tasks are instigated and processed in this domain. This in turn will invoke and utilise the internal architecture of the system, which is provided by the Kernel facilities. The kernel is the main storage and access controller, handling many of the filing and administration tasks, including distribution and transmission of information to remote GENIE-M hosts and applications. These domains combine to support the fundamental concepts underlying the GENIE-M system.

Section 8.2.1: The Kernel.

The kernel will supply tiles as requested by the application (via the server). Updated tiles are returned to the kernel when no longer required, or when a time-out mechanism indicates that the tile is unlikely to be accessed because it has been left unused for a considerable amount of time (perhaps one second or so). As the tiles are updated and altered, the kernel provides a historical or archiving function so that cross-referencing can be supported. Tiles within the system are always related by a master web structure that describes the overall organisation of the complete tile space. Any given application or user will have a localised copy of a web subset, which relates the tiles stored in the local workspace or 'Topological Tilestore'.

Tiles and their contents are never actually deleted. Instead, they would be relocated to increasingly 'remote' storage. Thus, a typical sequence would be RAM chip memory, magnetic disk memory, magnetic tape memory, and possibly optical disks in the form of write-once, read-many (W.O.R.M.) drives.

The kernel is intended for connection to at least a local area network (L.A.N.), and possibly further afield. Accordingly, it would be directly responsible for connection to, and interaction with, other (remote) processing hosts. There is no architectural requirement for the kernel to be resident on any particular workstation or processor, the important point is to isolate the kernel *activities* so that the server and user-interface domains can be directed towards the more specific tasks required for the portrayal and manipulation of GENIE-M applications.

Section 8.2.2: The Server and User Interface.

The server and user interface domains represent the main components of a GENIE-M application. They perform all internal communication tasks in a manner transparent to the user(s), as well as using the kernel to manipulate and relate tiles in an organised fashion (see Figure 6). Each of the modules within the domains have a specific and clearly defined set of tasks to perform, depending on the actual application. Certain of the modules may perform tasks of an administrative as opposed to operational nature, in that they are capable of initiating actions - or responding to external real-time events - without requiring user directives; but the majority of the modules would only operate when they are given specific tasks to perform from the user or from higher modules.

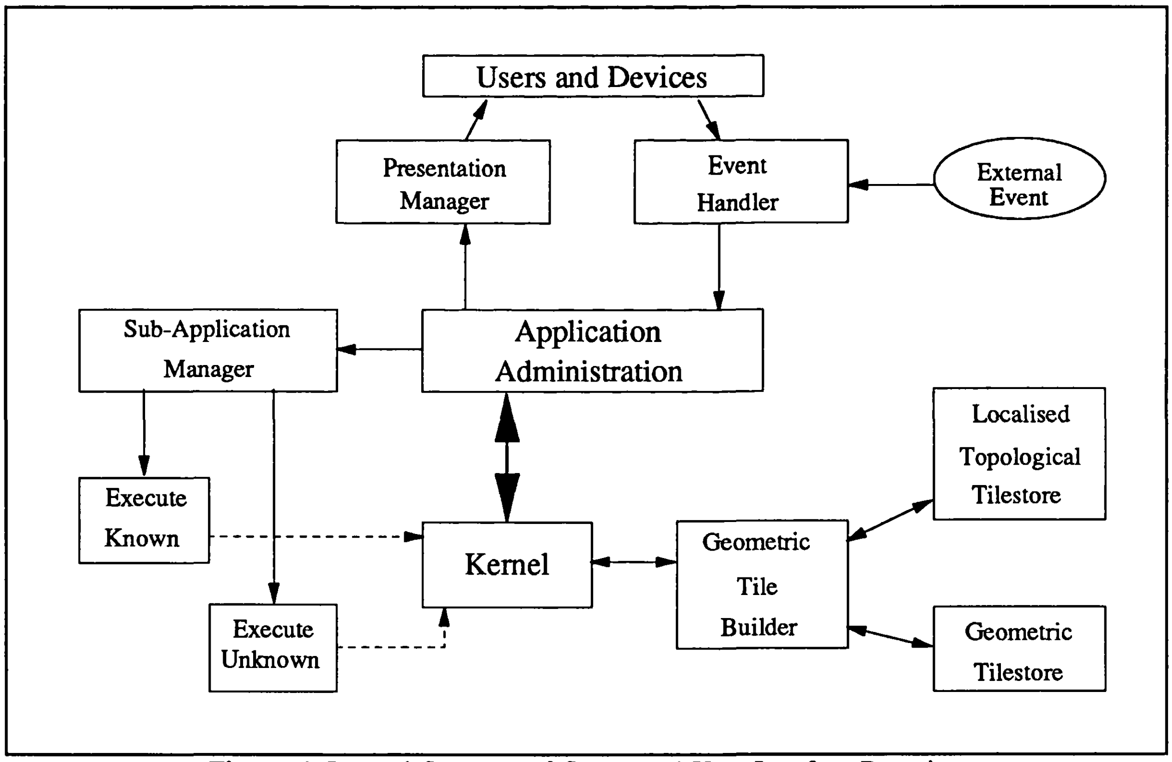


Figure 6: Internal Structure of Server and User Interface Domains.

Section 8.2.2.1: The Application Administrator.

The Application Administrator domain is responsible for receiving user or real-time event input and converting the commands and initiating circumstances into the appropriate directives to other modules to achieve the desired result or response. Given the complex nature of this domain, it is envisaged that it could be constructed using an intelligent or expert system, in order to be able to serve the users in a more relevant manner. The parameters of this domain will define precisely the application at hand, and the execution and responses specified in the construction of the functional components will determine the method of interaction. In practice, modules within this domain (along with those in the kernel domain) would be the first to be activated in order to request initiation of the supplementary support modules. Once activated, and all I/O channels have been constructed, modules in this domain can await user instructions. These are then interpreted and appropriate directives passed to other domains within the GENIE-M application architecture.

An application-aware multimedia editor would be mandatory for this domain, and would perform the actual editing of the contents of topological tiles.

Section 8.2.2.2: The Presentation Manager.

This domain has a clear knowledge of the presentation devices under its control. For every presentation 'window' or 'viewport', the manager will have an internal record of its architecture, current contents, relationship with other viewports, ownership, priority, and other details. The presentation information is usually supplied from the Application Administrator, using data from the kernel. Responses from users would be directed towards the event handling domain modules, since the presentation domain is responsible purely for implementing directives from within the system.

A primary aspect of the presentation manager tasks will be the presentation of geometric tiles that have been constructed from the contents of the relevant geometrical tilestore. The geometric tiles have, in turn, been built up using the configuration and content data held in the topological tilestore. The geometric tiles are primarily intended for submission to the user via the presentation manager, and so are available to the presentation domain in their entirety. The advantage of this approach is that the actual presentation of data does not reflect the devices used, except at the final stage of computation.

However, the fact that the geometric tiles have been constructed and exist within the tile space means that they can themselves be referenced by other tiles using a linking mechanism similar to that used within hypertext systems. This would be useful for incorporating copies of the material as references into other electronic documents. Additionally, minor alterations to the presentation can be made without having to interrupt or reconfigure the remaining domains within the architecture. Instead, simple adjustments (such as scrolling or moving display windows on the presentation device) can be performed locally by the presentation manager.

Section 8.2.2.3: The Event Handler.

This domain handles the introduction of input events into the architecture. For the prototype system, such events would include keyboard or mouse activity, but for the generalised system the events would include voice, power-supply monitors, motion detectors, and any peripheral device that can generate an external interrupt requiring attention. The event handler domain modules ensure that the events are converted into a form that can be recognised and acted upon by the application administrator.

Section 8.2.2.4: The Sub-Application Manager.

This domain provides a mechanism by which the architecture of a GENIE-M application can be extended with new facilities or by incorporating existing tools. If the current application administrator is supplied with a directive that does not make sense within the current application, then the command can be passed to the sub-application manager to determine whether an external tool or module can recognise and process the directive. For external applications - which do not necessarily know about the GENIE-M architecture - modules in this domain can simply pass the requisite data to the external application, and wait for the output results to be obtained. In effect, the GENIE-M architecture will provide an operating environment for the external application.

This architectural provision also enables the use of tiles as executable objects, possibly in the form of programs or command files. Clearly, such a facility presents potential integrity risks with regard to the system validity, and so would be treated as external applications.

Section 8.2.2.5: The Geometric Tile Builder.

This domain is responsible for such tasks as storage and tile conversion. Topological tiles are referenced according to the generated web structure that relates them. Any request for tile material would be directed through or initiated from the application administrator, which will instruct modules in this domain to ready the appropriate topological tile for access. If not available, then a suitable error message would be generated.

The next stage of tile access requires the conversion of the topological tile contents into geometrical contents. This conversion process will be extremely application-specific, and may involve any combination of expert system, device driver and filter tools.

Section 8.2.2.6: The Tilestore Domains.

The tilestore domains are populated by controlling modules which perform physical storage and filing tasks, as directed by the Geometric Tile Building modules. The tilestore domain modules would have responsibility for keeping track of all the tiles currently in use within the system, as well as incorporating new tiles and disposing (archiving) outdated or unused tiles. As described in section 7.2, tiles are stored within a tile space; but for convenience tiles may be stored in different domains within the tile space, in order to help categorise them according to their type. This has the additional advantage that manipulation of tiles can be controlled according to ownership - privately owned tile would be stored in one domain, while publicly available tiles would be stored in a more accessible domain. The two main tilestore domains are as follows.

Section 8.2.2.6.1: The Localised Topological Tilestore.

This storage domain holds and provides access to the topological tiles, according to requests from the Application Administrator. Certain of the topological tiles will be specific to individual users, while others will be for multiple/general access by many users. In effect, individual users accessing this tilestore will have their own sub-space within the tilestore. In order to implement this facility, users would have a Privileged Access Descriptor (P.A.D.) entry within the administration storage of the tilestore, which is used to determine the validity of the user's request. Such a mechanism allows a single tile to be shared amongst several users, but with only one copy being held within the entire tilestore; the individual users will have their own P.A.D. in their own sub-space.

Section 8.2.2.6.2: The Geometrical Tilestore.

Modules within this domain perform very similar tasks to those performed within the Topological Tilestore domain: the major difference is that there is no need for users to have their own sub-areas - if the user was not permitted to access a topological tile, then a corresponding geometrical tile could not have been created. This tilestore holds the geometric tiles that are the result of formatting the topological tiles, and which will be manipulated by the presentation system. In practice, tiles within this tilestore can still be referenced by external tiles that are located within other tilestores, but for administrative purposes it is convenient to distinguish the tilestores.

Section 8.3: Hardware and Software Considerations.

As discussed in section 7.3, the separation of processing tasks into distinct components reflects the definition of individual processing modules, rather akin to the 'processes' on a multi-tasking computer architecture; an approach made more important by the inevitable real-time nature of certain data items (e.g. date, time, etc.), and the fact that many changes could occur concurrently within the information environment. Accordingly, the development of any second-generation computer-based information system must reflect the nature of the distinct modules by implementation using a truly multi-tasking computer system.

The initial development for this research work was carried out in the Computer Science Department at University College London. The department has an extensive range of resources, particularly with regard to machines suitable for real-time multi-tasking[58]. The main machine is a Pyramid computer, with additional support provided by VAX 11/750 machines, all running UNIX in a variety of forms. At the lower end of the computer spectrum, some Sun Workstations (running the SunOS implementation of UNIX) and several IBM-compatible Olivetti Personal Computers are available. The range of possible hosts therefore was greatly beneficial with regard to the development of multi-tasking components of the architecture, as a large number of experiments could be carried out to evaluate the nature of the multi-tasking and communications required by the multiple modules of the architecture.

An immediate result established was that the hosting machine architecture is directly relevant to the performance of multiple process activity. In general, if a computer is multi-tasking, it will attempt to 'load' a number of programs into its primary memory, and these are formed into active processes which can then be executed logically according to a given scheduling strategy. The number of processes which can be loaded at any one time depends on the amount of memory, the size of the processes, and the nature of the jobs being carried out by each of the processes. Naturally, evaluation of the behaviour of such systems can only be carried out in general terms, by taking statistical measurements. However, even without detailed analysis, it is clear that multiple, small tasks which function only on an occasional basis (when needed) are very likely to be temporarily 'swapped out' from computer's memory, resulting in a noticeable performance degradation when they are retrieved from the swap space [Madnick 83]. Consequently, the only way in which an architecture such as that proposed - which has many such processes - could operate feasibly is when each of the modules can be guaranteed to remain in memory for the maximum

58: A major reason for this is that the department has a special interest in computer communications research.

amount of time, with a high scheduling priority[59]. Such a property is normally associated with operating system processes, and those of system manager privileges. Naturally, this was not possible for the 'public' machines within the department, a problem which was not allayed until a small workstation became available for dedicated use.

The provision by the Science and Engineering Research Council (S.E.R.C.) of an I.C.L. Perq computer, running a version of UNIX called PNX was extremely helpful. The machine itself had a high-resolution monochrome monitor that enabled initial development work on a graphics front-end to the architecture. Unfortunately, however, the version of UNIX used as the basis of PNX was rather old, and as such had little in the way of support for communication between multiple processes. As a result, the early system development work had to be carried out in two separate forms. Modules that could be tested in a stand-alone fashion and those for use in the graphic front-end were developed on the Perq, while the kernel modules and options for communications routines were evaluated using the Pyramid and/or VAX computers.

The initial development work had to be carried out using UNIX (or UNIX-like) hosts, because these were the only multi-tasking machines readily available. However, serious consideration was given to alternative architectures for development work, including using IBM PC hosts for low-end implementations of the architecture, and alternative operating systems for high-level but UNIX-independent implementation, the most obvious example being VMS. However, the hosting operating system chosen was UNIX primarily for convenience in that it was provided across multiple machine architectures; but also because it clearly had support for the features required in order to develop the prototype system ready for evaluation.

The decision was later to prove correct following the relocation of the research project to the site of the Department of Information Systems at the London School of Economics and Political Science. The department had procured two of the latest Sun Workstations for the research, which supported not only a superb development

59: Such modules are said to be 'sticky', because they remain in the computer's memory even if they are not currently running.

environment for both graphics and system architecture work, but also one of the most recent versions of UNIX with numerous extensions, especially in the area of inter-process communications (see section 9.2).

Nevertheless, the work carried out using the Perq and Pyramid/VAX machines was invaluable in the pre-prototype development, as it enabled the study of such components as module communication when modules were situated on the same or on different hosts[60]. Such an approach proved highly successful, in that when the modules were ported to the Sun environment, no modifications were required for successful compilation and operation, although the modules have since been 'fine-tuned' to take advantage of Sun-specific features and improve overall performance.

The choice of development language was inevitably limited by those available on the hosting machines. The one common language was 'C', and so this was chosen for the purposes of primary development. However, with the exception of certain aspects of module communication - which was closely dependant upon the facilities available with the UNIX environment - the majority of the prototype code could be written in any language of choice, as long as it fitted within the general architecture. In accordance with the module philosophy, as long as any given module could function using tiles, and communicate with any necessary external modules, then the exact operation and even nature of the module was completely irrelevant. This relaxation of constraints upon module development immediately means that the most efficient or familiar language could be used for any specific module, but one of the inconveniences would be that it would not be possible to specify a limit on the execution environment of any module. A module must be able to operate within what would be perceived as a virtual machine, and so not be required to take into account the available memory

60: Experiments concerning module communication between two heterogenous architecture computers (Pyramid and VAX) were extremely enlightening. In particular, it was found that the then state-of-the-art UNIX implementation had certain limitations. Quoting from the documentation [UCL 83]:

> '[Some sections] are an experimental part of the operating system more subject to change and prone to bugs than other parts.'

In practice, it was found that under certain program conditions, code would be executed that would literally halt a Pyramid computer. Normally, this should be completely impossible for an ordinary user applications program. Needless to say, this was an unpopular experiment with other users, and so not repeated.

or resources. This is one of the reasons why a truly parallel architecture host machine would be preferable for a production version of the GENIE-M system.

There are certain other considerations, however, particularly with regard to the presentation of data to the user and the range of devices that can be attached. Although UNIX hosts are ideally suited to the development of the multi-tasking nature of the proposed architecture, the hosts are often limited with regard to the range of presentation options. Specifically, it is true to say that the only standard mechanism for presenting data from a UNIX machine is through a text-only V.D.U.

Certain limited exceptions do exist (for example, the Sun View windowing environment), but there is almost no predominant standard for graphic presentation of data. One possible exception that is gaining support amongst UNIX system developers concerns the X-Windows interface, which is intended to provide a common graphics tool-kit for applications programs. An alternative that may prove competitive is the Adobe toolkit called Display PostScript, a screen-based version of the definitive printer-orientated Page Description Language PostScript.

In the absence of a clear choice for display generation, it seemed sensible (and a beneficial use of the modular architecture of the information system model) to relocate experimental versions of display modules onto machines or devices other than the UNIX hosts. Accordingly, consideration was given to developing a prototype system that could be controlled either directly via the UNIX hosts, or through a simpler control system running on an independent personal computer that supported high-resolution graphics. Two forms of direct interaction with the UNIX hosts were defined:

1) A powerful and detailed graphics environment for user interaction, in accordance with the ideals specified in chapter 7.

2) An elementary command-based interface that offers a ‘universal’ portability, but which is achieved by omitting graphics enhancements.

For the second case, the UNIX hosts would be controlled by the commands entered at the elementary interface, and respond or present results according to the application configuration. Two configurations were designed. The first was to use the GENIE-M generator architecture to present data to an output-only graphics device, called the IKON Pixel Engine, a graphics processor which was successfully connected to the Sun Workstations using a serial cable and serial device-driver software. This would allow demonstrations of how a system module generated by GENIE-M could construct a graphic representation of the data presentation, which could then be displayed for the user to interpret.

The second configuration required the combining of the second interface with the personal computer control system - in effect turning the PC into a semi-intelligent[61] front-end host. This second configuration was embodied in the development goal of a simple system which was generated using the prototype system developed by the author (see Figure 7). An advantage of this configuration is that it would demonstrate very neatly many of the objectives of the GENIE-M computer-based information system generator. The vast majority of the complex and multi-task processing jobs could be carried out on a dedicated host, while the more straightforward and distinct tasks for data I/O could be carried on a dedicated front-end machine.

Section 8.4: Summary.

In this chapter, we have considered the overall analysis and design of the GENIE-M generator. An exploratory prototyping approach was used to specify the system, which was seen to be a sensible approach in the light of the changing environment being used for the final development. The architecture of the generator has been discussed in detail, since it is this - rather than the actual implementation - that is of key importance to the practical research.

61: Semi-intelligent because although the PC would in effect be under the control of the UNIX host with regard to the display of data, a complex program package would have to be running on the PC to provide the command processing and result reception/presentation.

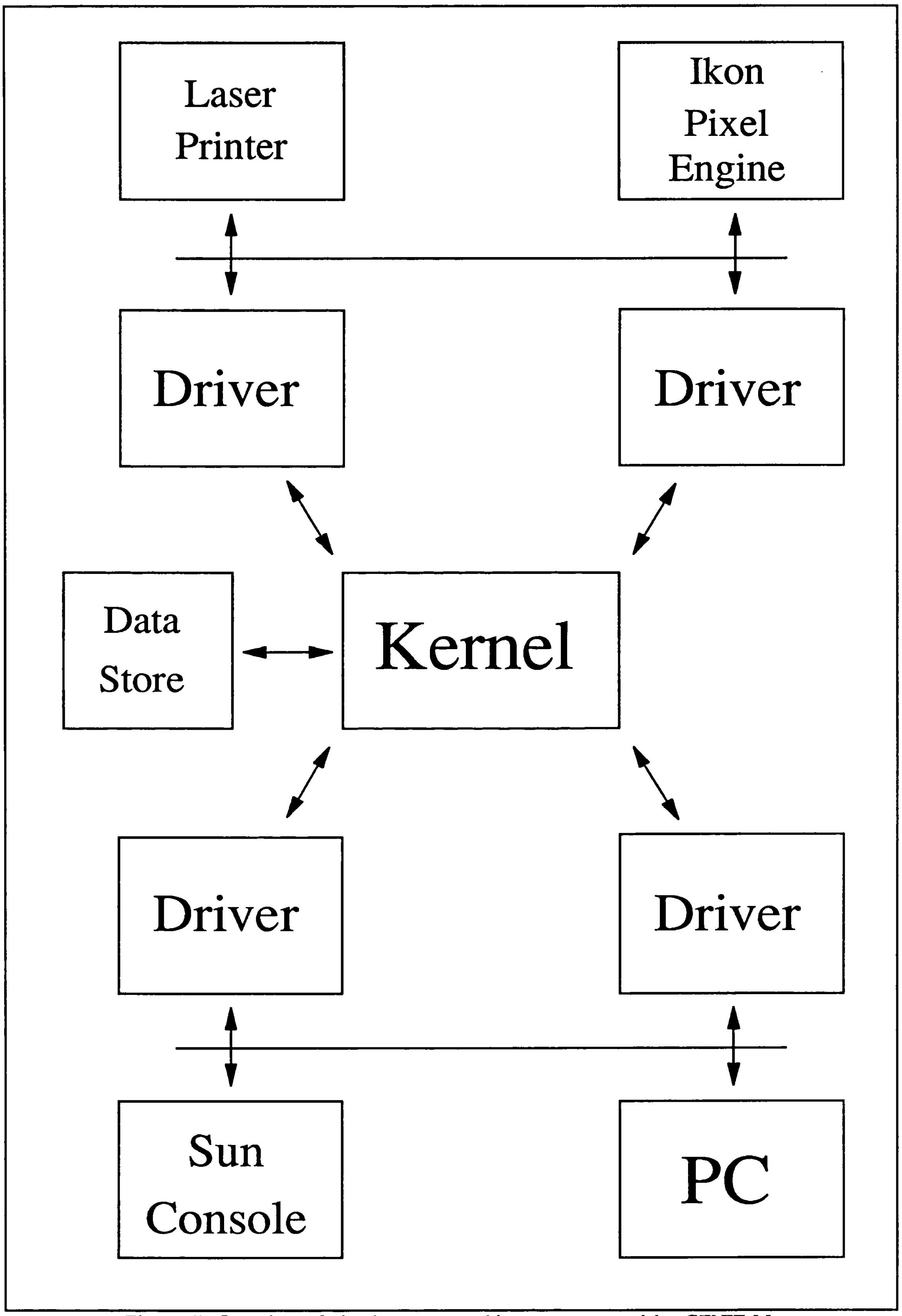


Figure 7: Overview of simple system architecture generated by GENIE-M.

The implementation itself is naturally influenced by the development environment, and where possible the constraints on design that resulted from the particular environment have been described. In the next chapter, we will consider the actual implementation of the generator, describing in detail the construction of the resources within the major generator domains.

Chapter 9.

Implementation of the Prototype System.

Section 9.1: Implementation Details.

As discussed in chapter 8, the implementation work was partitioned into two stages. In the first stage, a set of pre-prototype experiments were carried out, the results of which were used in determining the course of the second stage: developing the prototype generator system. The pre-prototype experiments were carried out using Pyramid and VAX machines at University College London, while prototype development was carried out primarily using the Sun Workstations at the L.S.E. This section of the report discusses the implementation details of the prototype system, taking into account the experimental and design work described in chapter 8. The implementation work built directly upon that carried out at U.C.L., but where additional points arise that were unique to the research at U.C.L., these will be identified.

Given the availability of a stable hardware platform in the form of the Sun Workstations, it was necessary to carry out a thorough evaluation of the extensions to UNIX that were offered by these machines, in the light of the experimental results obtained from the Pyramid and VAX machines. The initial implementation of UNIX offered was SunOS 3.2, with new releases of the operating system progressing during the development work from SunOS 3.4 to SunOS 4.0. The latter version introduced a completely new organisation for the file system on the host computer, but this had minimal effect for the implementation of the prototype system, since the design was as host-independent as possible. However, the reorganisation proved invaluable in identifying host architecture dependencies, which were noted so that for a production system, which would necessarily have to provide for potentially extreme variations in architecture detail in both hardware and software, the mechanism of storing files and data could be variable, especially when exchanging and communicating data between hosts.

The pre-prototype experiments were broken down into two distinct groups. The first concerned the communication mechanism between multiple independent modules that need not necessarily be resident on the same host. The second concerned the development of module architecture for handling the data tiles.

Section 9.1.1: Implementation Concepts.

For the following material, a number of terms are used which have a specific meaning in the context of this thesis.

A 'tree' is considered to be the internal representation of the current application. It is the internal construct which describes the information environment of an application, and models the data space within which tiles exist. Changing to a different application involves installing a different 'application tree'. In practice, several trees can exist for an application, and this mechanism is used for partitioning the tile space into the domains which are described in section 8.2.2.6.

A 'group' is a collection of tiles, which would normally be independent of each other. The use of the group concept allows the information environment to be manipulated while taking into account the relationships that exist between tiles. The prototype described in this chapter uses the group mechanism to provide a means of applying transformations to multiple tiles within a tile space.

The 'name' of a tile is simply the label by which it can be uniquely identified within the system. Any strategy can be used for naming, and for direct interaction with the system, a naming policy is used whereby tile names reflect their contents. Application modules are free to generate and use their own names.

A 'buffer' is a storage space used by modules for storing or retrieving blocks of data. The main buffer of any significance is `GENIE_buffer`, which is used for the exchange of data between modules. When a message is received, the contents are placed into a temporary analysis buffer while they are analysed, however this buffer is not normally available to users or modules.

The concept of the 'current' tile, tree or group simply refers to the tile, tree or group upon which operations will take effect, unless otherwise specified.

Section 9.2: Module Communication.

The UNIX Operating System (having been selected for reasons described in section 8.3) provides a number of mechanisms for communicating data between processes, with the range of mechanisms being dependent partly on the version of the Operating System being used. The first and only standard mechanism is that of the 'pipe'. This is a shared communications link that exists between two processes spawned from a single parent which carries out the `fork` system call[62]. Although the pipe mechanism is straightforward, and for many problems represents an ideal solution; it nevertheless has the major snag of requiring that any processes wishing to communicate via a pipe must share a common ancestor. This is particularly difficult to achieve if the processes are running on separate host computers. Accordingly, the pipe mechanism had to be discarded as being too limited in its facilities.

The limitations of pipes are so severe that certain implementations of UNIX have worked to provide alternatives to pipes. The result has been a number of alternatives, including the reasonably common concept of the 'named pipe', where pipes can be created and connected using names. However, more divergent implementation of UNIX have gone further, so that B.S.D. UNIX[63] offers 'sockets', while System V

62: The `fork` system call works on a running process in much the same was as a 'copy' command works on a stored file - a duplicate of the original is produced. In the case of the `fork`, any process which executes such a command causes the generation of an exact duplicate process - called a child - which inherits (amongst other things) access to all the currently open files and one end of a communications pipe. The original forking processes continues and is called the parent process.

63: The original UNIX Operating System was developed by Dennis Ritchie and Ken Thompson at AT&T Bell Laboratories on an unused PDP-11 [Ritchie 74]. It was not viewed as a commercially viable product, but was available to Universities and academic establishments at minimal costs, although without support. Development of UNIX continued in particular at the University of California at Berkeley, which unofficially took over the role of distributing UNIX to other academic establishments. UNIX quickly became more popular, and as students graduated and entered the commercial world they wanted to take UNIX with them. AT&T recognised the existence of the commercial market, and so released their own versions, with the most recent being called System V. The most recent Berkeley Software Distribution (B.S.D.) version is called 4.4 The differences between the two versions are the main background for

(continued...)

UNIX offers the combined resources of message streams, 'shared memory' and semaphores, as an implementation of the concepts proposed by Dijkstra [Dijkstra 68].

Sockets are analogous to hardware ports on a computer. Any process can establish (create) one or more sockets within the file system of the hosting machine. Any other process wishing to communicate with the original process simply 'connects' to the socket with a socket of its own, and thus establishes a link down which data can be communicated. In practice, data is transmitted by considering the socket to be a variation on the theme of a file handle, so that data can be conveyed by using the `read` or `write` system calls that are often used for ordinary file handling. The parameters used at the stage of the socket creation determine whether the data transmission is simplex, half-duplex or full duplex[64]. An extension to socket communication is the concept of 'domains'. Sockets which exist within the UNIX domain effectively exist on the same UNIX host, thus allowing processes on the same machine to communicate. Sockets which exist in the Internet domain exist on different hosts - and so the sockets actually require to have a hardware connection associated with them - and this enables communication between processes on different hosts[65].

The System V paradigm of messages allow independent processes to send formatted data streams to arbitrary processes. Having established a message mechanism within the file space, the process can transmit and receive messages that are formatted according to a user-chosen integer type [Bach 86].

The shared memory communication offered by UNIX System V is a more complex technique involving much more work on the part of the operating system. Two processes which require to communicate will effectively specify an area of main memory to which they both have access. This enables the processes to exchange data

63: (...continued)
the difficulty in developing not just a genuine UNIX standard, but also for such problems of designing a common graphics or user interface.

64: Simplex: One-way only; Half-duplex: Two-way but not at the same time; Full-duplex: Two-way and at the same time if desired.

65: Internet domain sockets are still very new forms of software technology, and it was during experimentation with these that the Pyramid crashes described earlier took place.

by reading or writing from the appropriate areas of the shared memory. A number of system calls have been implemented to facilitate this form of communication. The shared memory space cannot overlap any other regions within the virtual space of the machine, and there is no apparent way in which to use this mechanism between independent machines.

The semaphore mechanism is a well-understood implementation of Dijkstra's work on process rendezvous, allowing them to synchronise their actions. There are inevitable but recognised and understood difficulties involving semaphores, which include the possibility of a semaphore locking access to a resource and leaving it locked either by accident - following a system crash - or erroneously resulting from poor programming. In practice, the System V semaphore implementation allows the unattached semaphore information to be discarded in appropriate error situations.

Interestingly, while the SunOS implementation of UNIX is based primarily on the B.S.D. version, the most recent version follows an agreement with AT&T Laboratories such that the 'official' line of development for UNIX will assume AT&T software running on Sun workstation hardware. Accordingly, the more recent versions of SunOS (4.0 and 4.0.3) all support B.S.D. and AT&T extensions - although not simultaneously.

The implementation of UNIX at U.C.L. was the B.S.D. 4.2 version, and accordingly supported sockets rather than the System V options of messages, semaphores and shared memory. Consequently, it was the former means of inter-process communication that was adopted for implementing module communication in the prototype information system. An additional bonus of the socket technique is that is builds upon the established mechanism of input and output to ordinary files, and so should be easier to understand and extend.

One peculiarity that was encountered and has not as yet been resolved concerns the sequencing of data transmitted through sockets. When a socket is created, a number of connections can be left outstanding if a connection is already in progress. As soon as the current connection finishes, the next connection should be automatically made, and the invoking process released from its 'blocked' state. The number of outstanding connections can be defined when the socket is first created, and currently ranges from zero to five connections being kept in a blocked state. Within the UNIX domain for socket creation, certain protocols for the data exchange can be defined, one of which concerns the sequencing of the data messages[66].

For the prototype system, a stream connection mechanism was employed which consumes rather more machine resources but ensures a reliable and sequenced exchange of data. However, during the implementation of the module communication on the Sun Workstations, it was found that for some reason, the messages were arriving out of their correct order. An additional mechanism was introduced to try and eliminate the problem, in that a locking file was associated with any given socket. Only if the serving module was willing to accept a message would the lock file be 'opened'. Thus any module wanting to transmit would simply check the status of the lock file rather than waiting to try and establish a connection with a possibly busy socket. Hence communication simply could not occur unless the serving module was willing to receive the data.

Yet the errors in sequencing still occurred. When the number of outstanding connections was reduced to zero (so that any module wanting to communicate simply had to wait until the socket was free), the sequencing problem disappeared, although the elapsed time increased slightly as there were more delays. It is certainly possible that the problem was caused by errors in the module communication software, but the actual coding was as correct as possible, and in accordance with text-book examples such as they existed. One possibility is that modules might still be swapping out of the execution space while waiting for transmission or reception of data, and the delay in swapping in or out could possibly be connected with the sequencing errors. The matter is one that remains unresolved, and would merit closer investigation at a later date.

66: 'Stream' sockets provide sequential, pipe-like data exchange. 'Datagram' sockets provide datagram or packet-like data exchange. Stream sockets model connection-based virtual circuits, two-way byte streams with no record boundaries. Datagram sockets model datagrams in network communications. Additional work is required to deal with the problems of messages being lost, duplicated, or arriving out-of-order.

Given that the communication between modules had now been implemented in theory, the next problem was to decide on the nature of the messages being passed. Any module could theoretically receive messages from any other module at any time. This introduces the possibility of horrendous deadlock problems, but the establishment of a hierarchical structure to the module communication meant that a form of message flow could be imposed upon the design. A standard series of transmission primitives were identified as being the minimum required for module communications. Although more primitives could be implemented, the number is restricted to the smallest possible set that maintains full functionality. This was considered important so that optimising of the commands could be achieved more easily, and in order to reduce the problems of validation and verification. The actual data itself was readied for transmission by being placed into a buffer of known size (for the prototype system this is a contiguous block of 1000 bytes, called `GENIE_buffer`), which can then be communicated to the required destination. The transmission primitives are described in the remainder of this section.

Section 9.2.1: `GENIE_init();`

This is the initiating primitive for any module that wishes to have access to the information system communications structure. The only parameter required is the name by which this module desires to be addressed by other modules within the system. The jobs performed by the primitive include the establishment of the socket and locking mechanisms, and initialisation of the module's communications resources. For debugging purposes in the prototype version, an automatic log file is also created that records all data transmission involving the module. In a production version of the system, this log file would be used as an audit trail for - amongst other things - security, reliability and reconstruction purposes. The success or failure of this system call is described using the appropriate return code[67].

67: In order to match the UNIX host operating system more closely, the error codes returned are consistent with the existing UNIX error handling mechanism. Any system call which fails will return the error code -1, while a successful system call will return the error code 0. If an error has occurred (resulting in the -1 return), an error identifier will be returned using the external variable `GENIE_errno`. The complete list of error codes returned by the system appears in the Appendix.

Section 9.2.2: `GENIE_send();`

This module will attempt transmission of the data held in the buffer `GENIE_buffer` to the module named in the parameter of the call. If the desired destination does not exist, then an error code is returned. In any other case, the transmission mechanism will wait or 'block' until the destination is ready to receive a message. This is established by continually examining the status of the lock file. When the lock file is 'opened', a connection is made, and the data transmitted. As part of the process, any transmission is preceded by an announcement that identifies the name of the sending module. This is required for those serving modules which may require to be selective with regard to reception of messages.

A major problem concerns the use of the GENIE-M primitives in a distributed environment. Specifically, there is the probability that modules may be activated on remote machines, and so some mechanism is required whereby local modules can access and utilise the functions of remote modules. The implication is that a Remote Procedure Call (RPC) facility is required, but in practice this may not always be provided by the host operating system[68]. A discussion of this particular problem follows later in section 9.6.

Section 9.2.3: `GENIE_recv();`

This module is the main message reception primitive. Its first task is to clear out the message reception buffer, and then announce that the module is ready to receive messages. This is achieved by opening the lock file. The module can then wait until a connection is made from another module (at which point the lock file is closed), and the reception of a message takes place. The message is placed into the reception buffer (in the prototype system this is the 1000 byte `GENIE_buffer` block), ready for return to the invoking module. The lock file is closed throughout these tasks to show that the module is otherwise engaged, and is not listening for messages.

68: The SunOs implementation does provided RPC facilities, but they were deliberately excluded for the GENIE-M development.

Finally, the complete message is parsed into its component units. This is achieved by copying the contents of the buffer into a working store, which can then be scanned without risking damage to the original contents. The start of any constituent unit is defined as being the first occurrence of a non-white-space character[69] in the buffer since the scan started. As each component location is identified, a pointer is created that addresses the correct position in the message copy. This task is repeated until the entire buffer has been parsed. At the end, the complete message remains unmodified in the `GENIE_buffer` block, but an additional set of pointers (stored in an array called `GENIE_parsewords[]`) points to the start position of each 'word' within the copy of the buffer. In this way, it is much easier for a user module to decode any message received. In practice, the parsing process is also provided as a separate primitive (`GENIE_parse()`) which requires no parameters because it always operates using the contents of `GENIE_buffer`. This enables user modules to re-initialise the pointers to the original tokens, or alternatively to load a specific parse string into `GENIE_buffer` ready for parsing in special circumstances.

At the end of these tasks, the `GENIE_recv()` primitive finishes, and control passes back to the invoking module. As a final assistance, a special string variable `GENIE_from` is set to indicate the name of the module that sent the message which has just been received and parsed. This can be useful in certain circumstances, for example if the message is a request for data to be supplied, then the serving module must know where to direct the reply.

Section 9.2.4: `GENIE_recvfrom();`

There are many circumstances where a given module requires to be selective in its reception of messages from other modules. For example, if a module X requests some data from module Y, the X module is then interested only in receiving a response from module Y and not from any other module. Accordingly, the `GENIE_recvfrom()` primitive is offered, which takes as its sole parameter the name

69: Any character other than a space, tab, form-feed or similar position-changing character.

of the module from which a message is sought. This primitive invokes the fundamental `GENIE_recv()` primitive in order to await a message, and if it arrives from the specified module (this can be checked by examining the `GENIE_from` variable), then the receive is carried out exactly as described in section 9.2.3.

However, it is possible that a message might arrive from a module other than the desired source, and this poses the problem of what to do with the currently unwanted message. The solution was to implement a dynamic buffer, which invisibly stores any and all received messages that are not from the desired source. Later, when a general message is sought by invoking the ordinary `GENIE_recv()` primitive, the first step is in fact to scan the dynamic buffer to see if some temporarily ignored messages are outstanding. If so, they are removed from the dynamic buffer, parsed, and the results returned to the invoking module. If no messages are outstanding in the buffer, the module can announce its readiness to accept messages once again (by opening the lock file), and the receive process can continue as normal.

For the implementation of the prototype system, this mechanism works well. However, it is possible to envisage circumstances where a production system would be required to react to high-priority messages, for example during a system shutdown. Accordingly, the message passing scheme should be extended in a variety of ways, such as those described in section 10.5.

Section 9.2.5: `GENIE_stop();`

This primitive is used as the final message handling task for any communicating module. It will remove all the socket and lock files from the host system, and free all the unused space from the process, in preparation for process termination. One possible extension to this primitive may be to make it a one-way invocation, so that to use it will cause the call to finish without returning to the invoking process[70]. However, the experiments carried out using the prototype system have not produced evidence of any clear advantage to this.

70: In a fashion similar to the UNIX `_exit` call.

Section 9.3: Module and Tile Architecture.

As indicated in section 7.3, the specification for the information system architecture required that the contributing modules should be as independent as possible. The exact operation of the modules was considered to be of little relevance, with the sole condition that they performed the specified task(s). However, it was preferable that they should be able to communicate using a common understanding of tiles, so that the data could remain type-less as far as possible.

In an ideal implementation of a production system, the modules would actually communicate using tiles rather than messages - specifically by sharing common access to the tile storage space, but for actual implementation using multiple modules in a UNIX environment, this would require a move away from the B.S.D. UNIX socket communication techniques towards the shared memory options of System V UNIX. Much of the development work for the prototype system had to assume socket-based communication, and so some re-appraisal of the mechanics of data exchange between modules had to take place. The common understanding of tiles has been retained for the prototype, but the preference for actual communication using tiles has had to be deferred. Instead, tile data is transmitted using messages, so that modules can build up their own working copies of the tiles as required. Inevitably, this slows down the data exchange, although it has the advantage that modules processing data need only access the data and attributes that are relevant to their operation. Additionally, the logging or audit-trail tasks implemented for debugging purposes in the prototype is greatly facilitated by the messaging mechanisms. In a production system, the logging activities would be invaluable for performing system integrity and reconstruction tasks.

However, a significant problem with this form of serial data exchange concerns the need to ensure that all aspects of the tile content and status storage are maintained correctly. Specifically, which module has accountability for the 'current' version of a given tile? The solution involved the implementation of a tile administration module, as originally detailed in the early work on the architecture. This module has the overall responsibility for maintaining the working copy of the tile space which is relevant to the current application. Modules which need to access tiles are required to obtain the relevant tile material by passing a request to the application administrator, processing the tile contents as necessary, and finally request and update of the tile space by advising the application administrator of the changes. In practical terms, this has meant that the prototype system requires that modules obtain a working copy of a given tile, process it, then send a revised copy back to the administrator for inclusion as the new version of the tile within the tile space, subject to user privilege permissions.

The implementation of tile storage within the administrator module has proved a complex process, as it is immediately obvious that the tile space is necessarily a dynamic structure. Any given tile can hold a theoretically unlimited quantity of data, with correspondingly variable number of attributes associated with the data - thus maintaining the distinction between data and its type. Building a series of functions into the administration module has been perhaps the most complex task within the prototype development, as much of the work has necessitated testing to ensure consistency. It is in this category that the value of software engineering through prototyping became apparent, by imposing a ordered discipline on the development process. In many cases, the use of software engineering techniques has focused upon the culminating activity of producing working software. However, for this research, software engineering principles are advocated as being invaluable in the design stages which underlie the development philosophy. Experimentation was therefore essential to compare what was provided with what was desired, throughout the construction of the prototype. Functions had to be incorporated that supported multiple modules, and in several cases the modules could have similar purpose but contradictory implementation - for example, modules that presented data through an elementary text-only display, or complex graphical front-ends that had to be able to direct data to multiple display ports.

A family of tile and structural processing functions are provided within the prototype system, which combine to provide mechanisms for inter-module directive passing. The functions are invoked by messages which are normally constructed by the application administrator modules, since it is to this domain that users address their specific command requirements. The general format for a message would be:

```
COMMAND [SUB-TASK [goal]] [datum [datum [ ... ]]]
```

The following commands are some examples of those provided within the prototype system.

Section 9.3.1: ABORT

This command will be received by modules and treated as the shutdown instruction. Any module subservient to the current module will first be sent the ABORT command, before the current module brings its own operations to a halt. This command is required in order to provide an coherent termination mechanism.

Section 9.3.2: DEVICE

This command requires the following parameters:

```
DEVICE name type
```

Any module which provides a service - possibly by controlling or participating in the administration of a device - can 'announce' itself to the remainder of the architecture (or at the very least, to associated modules in the domain) by using this command. The name parameter is used to label this module, and is necessary for use with the GENIE_send() command described in section 9.2.2. The type parameter is for reference purposes, and would describe in very simple terms the general nature of the module. Common examples which are used within the prototype system are:

DISPLAY	used for describing a V.D.U.
INPUT	used for describing a general ASCII input device.
OUTPUT	used for describing a general ASCII output device.

This list is not exhaustive, and can be extended owing to the design of the architecture.

Section 9.3.3: GROUP

This family of commands is directed towards the manipulation of groups of topological tiles. The use of these commands facilitates the maintenance of certain of the geometric relationships that are depicted between tiles, rather than ignoring or discarding them. In practical terms, they allow the prototype system to apply commands to multiple entities. Only one group per user can exist at any one time, so that for multiple users the number of groups active within the system could be larger. A number of `GROUP` commands are provided, as described in the remainder of this section.

Section 9.3.3.1: `GROUP ADD`

This command has the form:

```
GROUP ADD name
```

The referenced (named) tile is added to the current group. If no name is given, then the currently active tile (see `TILE SELECT` in section 9.3.5.7) is added. If there is no currently active tile, then no addition can be performed. If this is the first tile in the group, then the group is initialised to consist only of the identified tile. Tile reference names are generated either by applications or directly by the user(s)[71]. Tiles can only be added once into a group. Any attempt at duplication is simply ignored, as it has no meaning.

Section 9.3.3.2: `GROUP DELETE`

This command has the form:

```
GROUP DELETE
```

All referenced tiles in the group are removed from the group (but *not* the tile space). The function of this command is to remove the relationship of the group, thus re-instating the independence of each of the tiles within the group.

71: UNIX provides a `tmpnam` system call for the generation of unique names. A bug in the Sun implementation means that the generation will cycle after the 17,576th invocation.

Section 9.3.3.3: `GROUP MOVE`

This command has the form:

```
GROUP MOVE x y z
```

All tiles which are members of the current group can be relocated within the tile space. This command will identify the relocation necessary to move the current tile (see the `TILE SELECT` in section 9.3.5.7), and apply the relocation values to all tiles within the group.

Section 9.3.3.4: `GROUP RMOVE`

This command has the form:

```
GROUP RMOVE x y z
```

All tiles which have been selected and added to the current group can be relocated within the tile space. This command will apply the specified *relative* relocation values to all tiles within the group.

Section 9.3.4: `SHOW`

This family of commands is intended purely for debugging purposes. It is provided as a command directive to examine interactively the current status of various aspects of the system. The output is directed to the primary output device, ignoring the normal tile presentation mechanism. Accordingly, the command set should not be considered part of the standard GENIE-M directives.

Section 9.3.4.1: `SHOW DEVICE`

This command has two possible forms:

```
SHOW DEVICE
SHOW DEVICES
```

The system will output a list of the currently known devices and their type onto the primary monitor. This provides a facility to ensure that all modules and device drivers are correctly operational within the system. The list has been constructed during the initial start-up of the system, and subsequently updated as modules and devices are introduced or withdrawn from the system. The list is verified by sending a test message to all modules or drivers supposedly present, which will then respond in some fashion. The nature of the response can generally be ignored, it is sufficient that a response was received in order to recognise that the module or device is present within the system.

Section 9.3.4.2: `SHOW GROUP`

This command has the form:

```
SHOW GROUP
```

This command will cause the system to present a list of the tiles currently within the active group, if any.

Section 9.3.4.3: `SHOW TILE`

This command has the form:

```
SHOW TILE
```

This command returns the complete set of data, parameters and attributes associated with the current tile. The data will include name, location and dimension within the tile space, and current content.

Section 9.3.4.4: `SHOW TREE`

This command has the form:

```
SHOW TREE
```

This command returns the complete set of tiles which comprise the current application tree. All data, parameters and attributes associated with each of the tiles within the tree are identified.

Section 9.3.5: `TILE`

The family of `TILE` commands is concerned with the direct manipulation of a specific tile, or some aspect of its circumstances. Certain of the commands are intended to support interaction with the host operating system, but following the intentions described in section 7.3 such that individual modules should be independent from each other, there is no specific provision for display or presentation. These latter tasks would be invoked according to the application itself, and in effect are accessed at a higher abstract level than the tile manipulation commands.

Section 9.3.5.1: `TILE CREATE`

This command has the form:

```
TILE CREATE name
```

This command is used to create an empty tile within the tile space. It is added to the current application tree, using the supplied `name`, but has undefined content, location and dimensions. Having been created, the tile is automatically selected as the current tile for subsequent tile manipulation tasks.

Section 9.3.5.2: `TILE GET`

This command family is supplied to provide tile interrogation facilities for the application, and comes in a number of forms.

Section 9.3.5.2.1: `TILE GET ATTRIBUTE`

This command has the form:

```
TILE GET ATTRIBUTE name
```

Tiles can have any number of attributes associated with them, for any purpose and containing any type of value. This command is used to establish the current value of the attribute identified by the `name` parameter. A series of messages will be returned to the requesting module, in the sequence as specified in Figure 8.

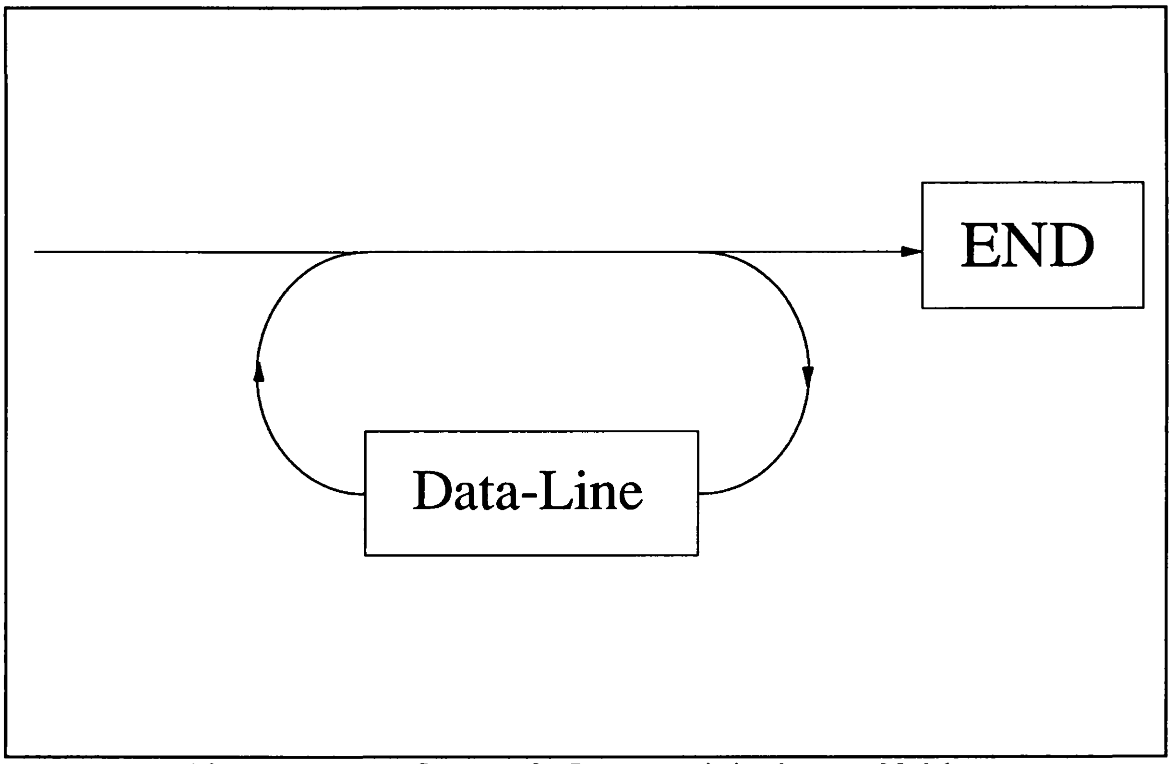


Figure 8: Message Sequence for Data transmission between Modules.

If there is no attribute identified by the given `name`, then the syntax provides for the return of the message `END` alone. However, if there *is* an attribute, then one or more data messages can be returned, in a similar fashion to that described in section 9.3.5.2.2 under `TILE GET DATA` and in Figure 9.

Section 9.3.5.2.2: `TILE GET DATA`

This command comes in the form:

```
TILE GET DATA
```

The complete set of data in the currently selected tile is returned to the requesting module. If there is no current tile, or no data, then only the `END` message is returned in accordance with the syntax diagram specified in Figure 8. Clearly, there is the possibility that the quantity of data to be transmitted will be very large.

For the prototype system, this is dealt with by automatically splitting the data messages into small packets, which must be reassembled by the receiving module. For a production version of the system, all participating modules would be able to access the tile space directly, rather than having to request copies of the tiles, and so the problem of segmenting the data would not arise.

In order for the receiving module to identify correctly the start and end locations of the data (and to allow for the possibility for unusual byte values in the data stream), the transmission of the data packets is such that each message has the form:

```
DATA ~<data>~
```

where the contents of the buffer between the ~ characters is assumed to be defined as bytes rather than as characters, and so may contain any value (see Figure 9). Should it be necessary to transmit the byte value of the ~ character itself, this can be achieved by using the sequence ~~ which is decoded as representing the single ~ character by the receiving module. To assist in the parsing of the received message, the `GENIE_parse()` function will ignore the contents of the analysis buffer between bracketing ~ characters.

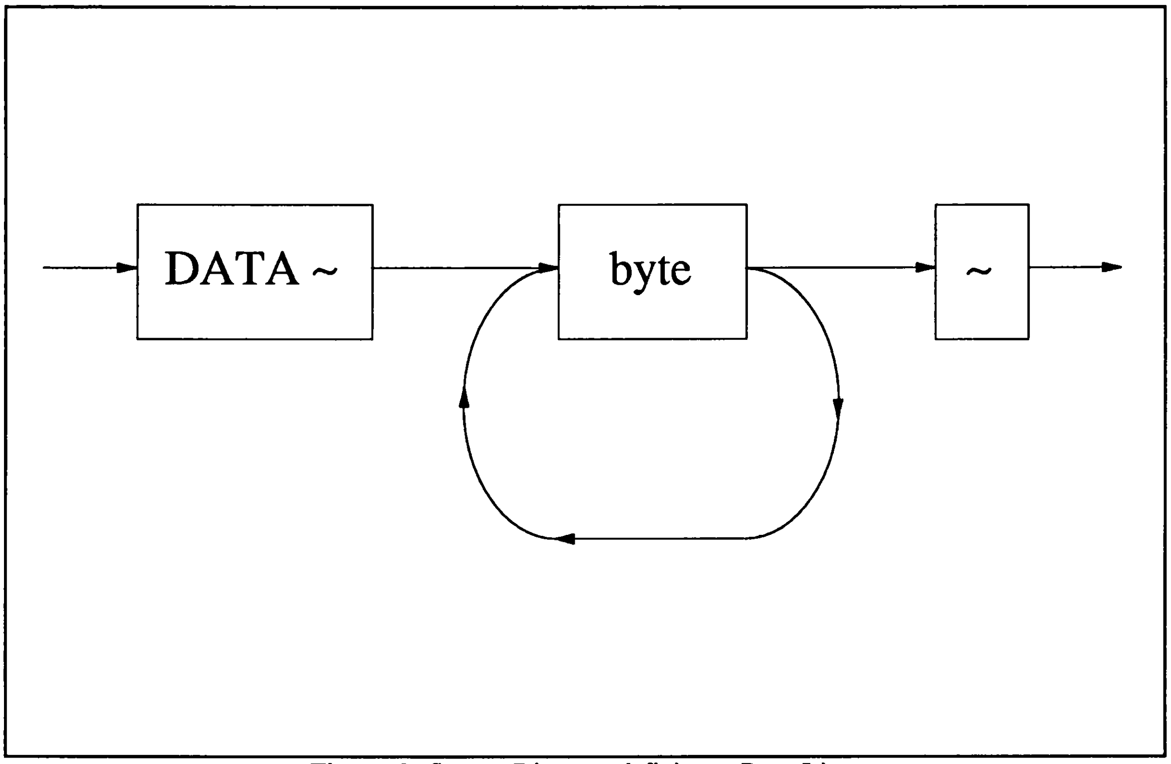


Figure 9: Syntax Diagram defining a Data Line.

Section 9.3.5.2.3: `TILE GET POSITION`

This command comes in the form:

```
TILE GET POSITION
```

The location of the currently selected tile is returned to the requesting module in the form:

```
DATA x y z
```

where `x`, `y`, and `z` are the location coordinates of the tile within the tile space. The `z` parameter refers to the 'plane' of the tile, and is useful for representing attributes of the tile that pertain to the overlap potential of geometric representation. No `END` message is returned, since only the single message would be required. In a production version of the system, this command would refer to locations within a potentially 'n'-dimensional tile space (according to the application), and so the number of parameters returned could be variable.

Section 9.3.5.2.4: `TILE GET SIZE`

This command comes in the form:

```
TILE GET SIZE
```

The dimensions of the currently selected tile is returned to the requesting module in the form:

```
DATA x y
```

where `x`, and `y` are the linear dimensions of the tile within the tile space. No `END` message is returned, since only the single message would be required.

Section 9.3.5.3: `TILE IDENTIFY`

This command has the form:

```
TILE IDENTIFY object
```

This command is used by modules to establish (identify) specific tiles according to the parameter specified in `object`. If no tile can be located, or the end of that search direction is encountered, then an empty result is returned. In all other cases, the unique identifier (suitable for use by a `TILE SELECT` command) would be returned as the sole value. A number of sub-forms of the command are provided.

Section 9.3.5.3.1: `TILE IDENTIFY NEXT`

This command has the form:

```
TILE IDENTIFY NEXT type direction
```

This complex command is used to return the identifier of the tile which most closely relates to the current tile (identified using `TILE SELECT`), according to the relationship specified in the parameters of the command. `type` is used to select between `INTERSECT`, `NEAREST` or `TREE`. An intersection is considered to occur if the two tiles overlap in any way when examined according to their current locations and dimensions within their respective planes. A near tile is one that is simply closer within the tile space than any other, in the direction specified. A `TREE` tile is simply the next tile that is stored according to the internal architecture of the tile space representation. This representation is highly implementation-specific, and should not be considered a standard feature of GENIE-M implementations. It is included specifically to facilitate debugging of application modules, by enabling a module to construct a complete list of all tiles within the tile space, regardless of their relative location within the tile space.

Direction is given in one of three forms: `UP`, `DOWN` or `PLANE`. The upwards direction is considered to be nearer to the root of the tile space (see `TILE IDENTIFY ROOT`), while the downwards direction is considered to be away from the root. The planar direction searches within the current plane of the tile space to find any matching tile. Searching for an intersection within the current plane has no meaning within the current definition of the geometric tile space, and so attempting to request a planar intersection is simply converted into a request for the nearest tile in the same plane. Specifying a direction with the `TREE` type will be ignored.

This command is perhaps the most complex of the function set described in this section. It is a fundamental command to be used by applications in searching for - and establishing the nature of - relationships. Use of the `TILE IDENTIFY ROOT` command enables applications to re-orientate themselves with respect to the tile space, and thereafter the relative nature of the tile space may be used to navigate the application. As tiles are identified and located, they can be modified and relocated according to the nature of the application.

Section 9.3.5.3.2: `TILE IDENTIFY ROOT`

This command has the form:

```
TILE IDENTIFY ROOT
```

The identifier for the tile nearest the root of the tile space - or application tree - is returned to the invoking module. This allows applications to return to the 'start' of the geometric representation.

Section 9.3.5.4: `TILE MOVE`

This command has the form:

```
TILE MOVE x y z
```

The location of the currently selected tile is set to the specified values, where `z` is the desired plane. The internal structure of the application tree will be automatically updated, and an error state produced if the resulting move produces intersecting tiles within the tile space. For the prototype system, the precise action taken on the announcement of the error state will depend on the application. Some applications may ignore the state, while others will take corrective or preventative action.

Section 9.3.5.5: `TILE READ`

This command comes in the form:

```
TILE READ name
```

This provides a highly simplistic interface between the GENIE-M system and the host environment. The file identified using `name` is read into the currently select tile, without performing any conversion of any kind. In practice, a reference is created to the original file within the host file system, so that optimum efficiency is maintained. Only if the tile contents are modified in any way is it necessary to actually incorporate the file contents into the tile. Since the command can be executed without intervention from the instructing module, the actual delay to the overall execution time of the application can be minimised. When the file contents are read in, an uncomplicated analysis is performed on the contents (assuming simple ASCII content) to establish some working dimensions for the tile. The algorithm used is similar to that employed by the UNIX `wc`[72] command and determines maximum values for the linear dimensions of the tile. These initial values can be modified using the `TILE SET SIZE` command, but are useful starting points in most cases.

Section 9.3.5.6: `TILE RMOVE`

This command comes in the form:

```
TILE RMOVE x y z
```

The location of the currently selected tile is moved relative to the tile space, according to the specified values, where `z` is change in the desired plane. This command is functionally very similar to the `TILE MOVE` command, and further relevant details can be found under the corresponding description (Section 9.3.5.4).

Section 9.3.5.7: `TILE SELECT`

This command has the form:

```
TILE SELECT name
```

The objective of this command is to provide a facility for selecting tiles within the tile space. The `name` of the tile must be unique, but can be in any form. Once selected, a tile can be manipulated or interrogated without requiring further identification. In effect, it becomes the default tile for subsequent tile-specific operations. This command is automatically performed when a tile is first created, to

72: The `wc` command is intended to 'word-count' an input file, producing counts of the number of lines, words and columns in the input, assuming simple ASCII files.

facilitate the ensuing configuration tasks. If the named tile cannot be located within the tile space, an error is returned. Until a tile is successfully selected or created, any following tile-specific commands will return an error.

Section 9.3.5.8: `TILE SET`

This command family is supplied to provide tile modification, update and configuration instruction facilities to the application. For convenience, a variation on the command family is provided in that whenever the sequence `TILE SET` is used, the text `TILE PUT` can also be applied. The command parsing process will automatically match the appropriate command to the sub-command. The family of commands comes in a number of forms, described in the remainder of this section.

Section 9.3.5.8.1: `TILE SET ATTRIBUTE`

This command comes in the form:

```
TILE SET ATTRIBUTE name value
```

Tiles can have any number of attributes associated with them, for any purpose and containing any kind of value. This command operates on the currently selected tile, and is used to set the current `value` of the attribute identified by the `name` parameter. If the parameter already exists, its old value is overwritten by the newly specified value. Attributes can never be deleted, as specifying no `value` will merely set the named attribute to the null value. The advantage of this mechanism is that certain attributes for tiles do not need to specify or represent values, it is sufficient to note merely that they exist.

Section 9.3.5.8.2: `TILE SET DATA`

This command comes in the form:

```
TILE SET DATA
```

This command announces that the complete set of data in the currently selected tile is to be updated by the instructing module. If there is no current tile, then an appropriate error message would be returned before any data can be exchanged. However, given that a tile is currently selected, and is eligible for updating (according to access permissions), then the instructing module proceeds to issue a number of `DATA` lines terminated by a single `END` message, as depicted in Figure 8. As with the `TILE GET DATA` message, there is the possibility that the set data will be very large. This is dealt with by automatically splitting the data messages into small packets, which are automatically reassembled by the receiving module. For a production version of the system, all participating modules would be able to access the tile space directly, rather than having to specifically update copies of the tiles, and so the problem of segmenting the data would not arise.

The same mechanism as described in the `TILE GET DATA` command is utilised in order for the receiving module to identify correctly the start and end locations of the multiple data packets (and to allow for the possibility for unusual byte values in the data stream).

Section 9.3.5.8.3: `TILE SET POSITION`

This command comes in the form:

```
TILE SET POSITION x y z
```

The location of the currently selected tile is specified by the instructing module where `x`, `y`, and `z` are the location coordinates of the tile within the tile space. The `z` parameter refers to the 'plane' of the tile, and is useful for representing attributes of the tile that pertain to the overlap potential of geometric representation. In a production version of the system, this command would specify locations within a potentially 'n'-dimensional tile space (according to the application), and so the number of parameters listed could be variable. This command functions in precisely the same manner as the `TILE MOVE` command listed in section 9.3.5.4, and is included to preserve orthogonality in the command set.

Section 9.3.5.8.4: `TILE SET SIZE`

This command comes in the form:

`TILE SET SIZE x y`

The dimensions of the currently selected tile is specified by the instructing module where `x`, and `y` are the linear dimensions of the tile within the tile space. This command is provided for those applications where modules will modify the tile contents. The modification may result in an adjust to the dimensions of the tiles, which must be notified to the internal tile administration. Certain commands will cause a size adjustment to occur automatically - see `TILE READ` in section 9.3.5.5.

Section 9.3.6: `TREE`

The TREE family of commands are provided to enable manipulation of the tile space as a whole. In practical terms, these commands facilitate the management of the particular application by administering to the electronic abstraction (the I.D.E.A.) *in toto*.

For the prototype system, the commands are limited by the assumption that tiles within the tile space are unlikely to separated into distinct domains. However, as described in section 7.2, in a production system the tile would be categorised according to their nature (topological or geometric), as well as according to their accessibility (private, group, or global access).

Section 9.3.6.1: `TREE LOAD`

This command comes in the form:

`TREE LOAD name`

The first task for this command is to ensure that the named electronic application document exists within the host file system. Having verified its existence, the original application document can be replaced into the host system (if it has been updated in any way), before the new application is loaded ready for access. After this command is completed, the internal application status of the entire system will be exactly as if the named application had not been interrupted.

Section 9.3.6.2: `TREE SAVE`

This command has the form:

```
TREE SAVE name
```

This command is provided primarily for backup purposes. Normally an application never needs to be explicitly stored, because any changes to the application are automatically recorded before a new application can be accessed. However, there are circumstances where an explicit save may be required. For example, in cases where system reliability is doubtful, or where a backup using an alternative storage device is required. The saving process uses a version mechanism which is hidden from the user, so that no save can erase previous information.

Use of this mechanism naturally leads to a steady increase in the use of the host file system storage space, and for the prototype system this is dealt with by running a support program to purge the older copies to an alternative store. For a production system, the purging mechanism would be incorporated explicitly on the assumption that a high-capacity tertiary storage device[73] would be available for automatic archiving.

Section 9.4: Coding - the Sun Workstation Development Environment.

Generally speaking, a development environment consists of an operating system and a collection of tools provided on a host machine. As far as possible, the development tools used in the production of the GENIE-M prototype system are expected to be available as standard on most UNIX machines, but there were two Sun-specific features that proved of assistance.

73: Such as a dedicated Write-Once-Read-Many (W.O.R.M.) optical disk drive. Current systems already offer in excess of 1 Gigabyte (1280 MByte formatted) storage capacity.

The first and by far the most important was the multiple window display provided by SunView. This enables the depiction of multiple UNIX 'sessions' on the same display, as if several terminals were attached and operational. During the development of communications between modules, the multiple window display was invaluable in providing debugging data. Normally, modules would not be required to output any status data, although one standard component was the incorporation of log files of all transactions through the communications schema.

This mechanism was intended primarily for 'post-event diagnosis', and was of little assistance during the actual execution of modules. The use of multiple windows meant that, for debugging purposes, a module could be temporarily assigned its own window, and thus direct status data to the screen. As an example of the benefit of this, the sequencing problem referred to above (during the discussion of data communication) was quickly detected by simply echoing received messages onto the screen.

The second, and less-often used resource was an enhancement of the standard UNIX debugging tool `dbx`. The Sun-enhanced program `dbxtool` provides a useful way to examine visually problems which arise with sections of code, particularly when unexpected UNIX core dumps[74] have been produced[75].

During production of the code, the multiple source programs went through numerous revisions, partly for error-correction, partly for inclusion of enhancements. In order to administer the evolution of code, most UNIX hosts provide a Source Code Control System (S.C.C.S.) This is an extensive set of utility programs that are used to maintain a record of program (source code) changes, and to support the reconstruction of any previous version at any time. This is enabled by the production of 'deltas', which are records of the line-by-line differences associated with a given version of the file. One of the more useful aspects of the S.C.C.S. resource is the

74: A core dump is a 'snapshot' of the process status at the exact moment that a fatal error occurred, and is invaluable in post-mortem debugging.

75: The UNIX operating system can be rather uninformative with regard to fatal program errors. By far the most common error message encountered is the singularly bland 'Segmentation fault (core dumped)'.

facility to include comments with each of the revisions, identifying why and how particular changes were made.

The multiple modules present and requisite for the development of the prototype GENIE-M generator represented a potentially complex problem with regard to the multiple compilation tasks. This was aided by the use of the `make` tool, which takes instructions from a '`Makefile`' that guides the construction of multiple and closely associated programs.

Section 9.5: Tile Space Storage Mechanisms.

In an idealised model of a tile space, which provides storage for tiles used during the processing tasks within the information environment, data storage should be based upon random-access techniques. This would permit multiple modules and processes to access the data in a non-sequential manner, according to the directives supplied by the user(s) or higher-level application modules. Inevitably, random access implementations of data storage are affected directly by the operating system, and indeed it is this aspect that provides the greatest restraint upon the portability of the internal environmental architecture. The kernel of the tile storage system is responsible for controlling the actions of processes in their tile manipulation tasks, and consequently it co-ordinates any accesses to data storage. The kernel is implemented in the prototype architecture by a single 'matrix' module.

Section 9.5.1: The 'matrix' module.

This module has two major functions. Firstly, it is responsible for building a three-dimensional matrix-like structure to represent the data space - and as such supports the primary tasks required within the core domain of the architecture. A three-dimensional tree is constructed (using generalised algorithms which could be easily extended for use in 'n'-dimensions), and this tree depicts the various tiles and hooks into the file system for physical storage of tile data. This allows a given information document to be described using a tree, and consequently alternative applications can be accessed by installing new trees. The trees themselves provide tile reference mechanisms which permit tiles to be shared across multiple applications and many users.

A disadvantage of this approach is that there is an overhead for the processing machine in providing support for the tree representation. Ideally, the tile storage could be more object-orientated, so that access and updating could be performed at a higher abstract level, rather than requiring the current machine and file-handling dependencies.

The second major task of the matrix module in the prototype system is the administration and provision of the internal GENIE-M commands. All directives intended for accessing and controlling given aspects of the application within the experimental system are directed towards the matrix module using the `GENIE_send()` command. The matrix module then identifies the nature of the directive and passes the instruction to the corresponding module within the system. This activity is performed automatically. Commands which are not understood by the matrix module can be offered to all known modules within the application. The modules can then respond according to whether the command is recognised or not. If the command is recognised, the matrix module can build a list of where that command should be directed if it is required in future tasks.

Section 9.6: GENIE-M and Distributed Systems.

At no point does any implementation of GENIE-M require a specific host or operating system architecture. To do so would be to impose constraints upon a system that is specifically intended to be generalised. One of the more complicated areas of GENIE-M concerns the inter-communication between modules on distributed systems. There is no requirement for any module to be resident on any particular computer within a given GENIE-M application, and accordingly, it makes sense for modules to be invoked on computers to which they are better suited. Thus graphic modules might be closely associated with computers enhanced for Computer Aided Design (C.A.D.) work, while computationally-intensive modules might require computers with numeric processor enhancement or floating-point accelerators.

The problem arises that a module may require the services of any other module, but will not - in the first instance - 'know' where in the distributed system the module is to be located. Certain operating systems do offer a form of Remote Procedure Call, where processes can access resources on other hosts, but such a facility is often highly host- or operating system specific. With a distributed UNIX implementation of GENIE-M, there is also the inevitable problem of multiple 'sessions' being activated, such that a single user can be 'logged-in' to the same machine, using the same directory structure, but with each session being completely independent from the other. Thus, an invocation of a GENIE-M module within one session cannot normally be accessed by any other session on the same or any other computer.

As a solution, the `GENIE_send()` command has been enhanced to allow for the possibility that communication is required with a module that is not present within the current session. The parameter for `GENIE_send()` requests that a message be transmitted to the named module. However, if the module does not appear within the current work-space, then normally transmission could not be implemented. This can cause peculiar effects if one user is accessing a high-level module at the root level of a UNIX machine - thus causing the creation of an access point for the module within the UNIX file system - yet another user cannot connect to that access point because it is not considered to be 'live' from within the current session.

In order to deal with this problem, `GENIE_send()` has been modified to test for the presence of the destination module as a live access point within the current processing session. If no such access point can be located, then the module is deemed to be inactive. However, rather than simply abandoning the transmission, `GENIE_send()` will attempt to pass the transmission through a gateway module in order to access other sessions that may be in progress on remote - or even local - machines. Two gateway modules are provided, `gatein` and `gateout`. The former module 'listens' on a hardware communications port for any data being transmitted from an external session. The data will have been configured for inter-session communication, and will be parsed to identify the desired destination module. As before, a test is carried out to determine whether the requisite module exists and is active within the current session, and either the message will be rejected or else it will be successfully forwarded to the desired destination.

The `gateout` module is automatically addressed if `GENIE_send()` fails to locate the desired destination module. This module identifies the destination of the transmission, to determine where the message should be sent. If no known module exists, `gateout` will offer the message to any and all `gatein` modules active within any session of the host machines anywhere within the distributed system. If a willing recipient is located, the destination address is recorded for future use, and the message forwarded as required. If no recipient is located, the message will be returned to its sender with the appropriate notification.

The implementation of this mechanism is essential in supporting the use of GENIE-M within a distributed environment, particularly since the actual provision of the resource must be as host independent as possible. A number of check mechanisms have been implemented in order to minimise the danger of an inter-machine loop situation being created, yet at the same time the activation of this mechanism should be as transparent as possible in order to facilitate the straightforward nature of the GENIE-M implementation.

Section 9.7: Summary.

In this chapter, we have considered in some detail the technicalities of how the GENIE-M prototype was constructed. The prototype system uses the limited set of commands described in section 9.3 with a high degree of success, and was suitable for generating a demonstration system which suggests that the concepts upon which GENIE-M is based have some degree of validity. In particular, the architecture has been constructed upon the combined principles of minimal complexity with maximum extendability. Additions to a computer-based information system of this form can be provided by the simple expedient of incorporating the corresponding module into the information environment. The command recognition process and message-forwarding facility detailed in section 9.6 enables the extensions to incorporated rapidly.

Navigation through the system depends in part upon the application being able to identify the relationships that exist between the tiles. Support for this requirement is provided by the `TILE IDENTIFY` family of commands, which can be used by application modules to build up a complete working record of all tiles (and their inter-relationships) for any given tile space - and hence for any 'unfamiliar' application.

Reference has been made to production versions of the system. This would entail extending and improving certain aspects of the prototype with regard to storage capacity and improving the speed of the intercommunication, but in practice this could be achieved by a process of replacing modules with implementations of higher functionality. The commands described in section 9.3 are intended primarily for use by application modules, but they are also suitable for controlling an application by direct instruction.

Chapter 10.

Conclusions.

Section 10.1: Introduction.

This thesis has described the development of an architecture which uses an innovative geometric concept as a basis for proposing an enhancement for the development of computer-based information systems. The nature of the enhancement centres upon the description and construction of extensive manipulation models called Information Environments. These are computer-based representations of all the key components which contribute to and participate in a given set of information processing tasks for particular applications.

The derivation of a specification for the Information Environment concept has resulted from a wide-ranging examination and reappraisal of many aspects of computer science; considering in particular two main fields. Firstly, the use of data structures in the depiction of geometric relationships; and secondly, the construction - or generation - of application systems using the resulting fundamental components. The construction of systems which utilise the geometric paradigm should be facilitated, and the systems themselves should benefit from the additional capabilities that the geometric representations appear to offer in terms of problem representation and solution implementation.

In this chapter, we reassess all of the research work in a final analysis. This is felt to be essential in view of the diversity and number of topics that have fallen under scrutiny, and it is considered important to recognise the overall contribution of the independent but related and interactive nature of much of the material. It is also hoped that this commentary on the work should be of assistance in identifying and inspiring further avenues of research.

Section 10.2: Review of the GENIE-M Philosophy.

A common problem with many application systems based on existing architectures and development strategies is that, traditionally, they focus upon very specific problem areas or tasks. This often has the consequence that only limited consideration is given to how the tools constructed and supplied within the system might cooperate in complementing each other and alternative application systems within the human operating environment. As a result, these focused systems often impose a set of constraining patterns or ideas on the users, possibly through the imposition of a set of limited operational techniques. Indeed, the traditional emphasis of data processing - which emphasises the technology - perhaps indicates that the use of a computer-based information system itself imposes the pattern and manner of operation.

More recently, systems have been developed with the intention of providing more consistent and cooperative tools; but these often seem to concede performance elsewhere, for example in terms of reduced functionality, ease-of-use or expandability. The underlying model which is used in the construction of the focused systems is usually derived directly from the application under consideration, and consequently the opportunity for use or re-use of components from previous work is rather less than might be desirable [Sommerville 89].

A more generalised approach would seem to be that the system should support the users in ways that reflect rather than hinder their personal preferences and natural skills. The tools used for performing the tasks should therefore be more compliant with the needs and circumstances of the application, and also build on the strengths of other tools. The recent trends towards more advanced and integrated tasks of greatly varying nature, such as those that would be performed during multimedia information processing [McCandless 90], further justifies the move towards comprehensive models of the working environments. These environment models are envisaged as a means of enabling a more detailed representation of the diverse components that contribute to data processing tasks, with the aim of assisting in the advancement of computer-aided information handling activities.

These requirements for improving computer-based information systems suggest that provision must be made for alternative approaches in preference to purely objective-orientated design and development. Specifically, the need for greater productivity in information processing activities means that techniques must be derived for more considered *overall* strategies for the development of tools and systems. A corollary of this is that greater recognition must be given to the overall circumstances of the tasks being performed, and consequently the nature of operational environment is a factor that should be incorporated more precisely within the application development model.

It is therefore suggested that in order to achieve the desired enhancements to information systems, design and development strategies should from the beginning have two major goals: firstly that of strengthening the support for the users in performing their tasks of work, and in constructing their own patterns and ideas. Secondly, the underlying architecture should be considerably more portable and generalised in order to support not only the flexibility of the system but also ease the development process.

Thus, a major emphasis of the research work described here has been the production of an architecture that permits a wider range of more flexible application activities. This is achieved by the construction of a totally new computer-based representation of the information processing environment. The work on GENIE-M has suggested that using a geometrical representation as a foundation for some of the data representation and manipulation may provide the necessary computer-moderated support. Accordingly, the use of geometry may be an appropriate technique to achieve the higher objective of a more flexible architecture.

The use of a geometric architecture within computer-based information systems makes provision for the required enhancement by incorporating a diagrammatic aspect to the data representation and manipulation activities. Rather than imposing a potentially complex and mathematically intensive set of rules and regulations upon the - possibly 'computer-naive' - users, a pictorial approach is used not only for problem representation but also for the architecture used to implement the solution. The use of geometry should therefore be of value in guiding efforts towards the construction, combination and re-use of appropriate tools for implementing the solution to the identified tasks. At the same time, it enables the representation of a variety of additional relationship expressions that might not otherwise be so easily expressed.

Using these concepts, a model of computer-based representation of information systems was proposed, which has been deliberately extended beyond other existing models in order to recognise the existence and benefits of multiple users and peripheral devices within the application environment.

In constructing the GENIE-M prototype, we are trying to provide an experimental testbed upon which the suggested geometric framework can be evaluated. This is achieved by constructing actual components, which can then be placed and configured according to requirement. The internal architecture upon which GENIE-M functions is intended to provide simplicity and consistency in accessing and manipulating resources and facilities within the environment, in a manner similar to that provided by operating systems in relation to the underlying hardware. An internal processing structure is constructed for the geometric representation which uses an innovative tile-orientated data concept. The tile mechanism provides an appropriate means of depicting the enhanced geometry and relational orchestration of the data being managed.

A unified command mechanism provided by the prototype system results in a user-understandable instruction set for controlling the operation of the multimedia environment. The overall simplicity of the architecture which underlies GENIE-M has been deliberate, and indeed reflects the conclusions reached by the development team working on the KMS hypermedia system (described in section 3.4). End-users would be able to tailor their own systems by locally redefining specific application modules or tiles, knowing that compatibility with the lower levels remains intact.

However, it is recognised that the nature of the concept makes it unsuitable for immediate implementation or configuration by inexperienced users: in its current realisation it is not a form of 'Lego' kit for information systems. Instead it should be viewed as a mechanism to aid in establishing an understanding and subsequently producing a representation or description of problems. It is important to recognise that for each instance of a new problem, new mechanisms or tools must be generated from first principles. However, as each mechanism is completed, the underlying architecture promotes its reuse and reapplicability in alternative circumstances with minimal reconfiguration.

A major thrust of the author's work has been the development and implementation of the innovative internal system structures, but it is obvious that in order to communicate with the outside world, computer-based information systems generated from a GENIE-M system would require access to a range of input and output (control and presentation) devices. The prototype system used in generating the demonstration system to be described in section 10.3 has an input and output control mechanism that is primarily text-orientated. The decision was taken by the author to implement access to the system via a text-based front end in order to concentrate more effort on the internal mechanisms.

However, the nature of the textual commands (as described in section 9.3) are the result of design intentions to ensure suitability for use by more powerful graphic front-end packages. More advanced systems would therefore make use of the enhanced detail and power provided by the graphical front-end packages, thus utilising the facility to employ additional applications modules and associated peripheral devices within the information processing environment.

An important section of the work concerns the software engineering approach used. A concurrent goal throughout the research was to experiment with the use of evolutionary prototyping as a technique for development of the concept and the implementation. This method has actually been promoted by the circumstances of the development, and as pointed out in [Ince 89], is 'a useful way to develop software'.

The considerable amount of thought and design work that has gone into the development of the concepts that GENIE-M embodies appears to be justifiable. Much of the time was spent looking at current system objectives in order to develop fully an understanding of what the specific objectives of this research should be, and only then could a fundamental set of specifications be derived. Some of the results from this early work have been presented at conferences in order to publicise them and invite commentary from professional researchers and system developers [Angell 87 & Warman 87]. The various reactions were very helpful in strengthening otherwise weak areas, and in identifying points which needed further clarification or development.

Section 10.3: The Demonstration System.

In order to provide a realistic demonstration of the concepts behind the geometric architecture of GENIE-M, a specific configuration has been constructed to illustrate the major features of the work. Borrowing from the lecture-support idea suggested in section 7.5.3, the system is used to construct an application that will be used for demonstrating itself. A single presenter will be provided with a small control environment running on a personal computer, which will be used to direct the proceedings. A script of the presentation will be provided, with key aspects highlighted and echoed onto large-scale 'repeating' displays for the audience.

Graphical information will be presented using two devices, a purely PC-based display, and a high-quality graphic device. Hard-copy of the relevant notes would also be provided automatically through an attached printer driver. The consistency of the underlying architecture will further be depicted in two forms; firstly by the use of different languages in constructing some of the contributory components, and secondly by supporting the operation of an external (non-GENIE-M specific) application.

The agenda for the demonstration will consist of an introduction to the GENIE-M philosophy and architecture, with diagrams to illustrate the tile and tile-space concepts. The use of drivers within the system will be shown with modest examples at the relevant stages of the proceedings.

It is planned that by using the system to construct a demonstration of itself, the feasibility of using geometry to enhance computer-based information systems will be clearly illustrated. The key feature is the fact that the demonstration will be constructed upon a completely consistent underlying architecture, such that reconfiguration is an uncomplicated process.

Section 10.4: Successes of the Work.

It is believed that there are certain application areas in which systems could benefit from the incorporation of some of the ideas. In research fields especially, GENIE-M suggests a number of mechanisms which may be of assistance for solving current problems. The multi-tasking nature of the architecture means that GENIE-M systems may be of value in the development of future parallel machines architectures, where multiple processors (rather than simulations of multiple processors by a single C.P.U. implementing some form of time-sharing mechanism) will be provided physically within the hardware. Indeed, the design for GENIE-M has recognised from the outset that implementation on currently available hardware is unlikely to produce fast or efficient systems. However, the design has been guided by the nature of resources that will probably be provided with the next generations of hardware. In the meantime, current implementations enable the design to be evaluated and tested in advance of the more suitable technology to come.

An example of a simpler software-orientated application that is practical at the moment would be a notebook, whereby a tile could hold text information, while another tile could hold graphic information entered via mouse, keyboard or light-pen/stylus. Any access to one or the other tile would automatically call up the other tile(s), in order to retain the relationship that is implied. A more complex application is the lecture support environment, which must take into account a number of apparently rudimentary objectives in terms of presentation requirements, yet maintain the application for a number of otherwise incompatible devices. A complete and structured script would be generated that includes not just textual material, but also graphics, pictures and other support elements. Each of these components is combined within the architecture to form single application 'documents', which can subsequently be used and re-used as required.

With reference to more advanced applications, the fields of multimedia and hypermedia systems would seem to be prime candidates for GENIE-M development. As discussed in section 3.4, Halasz raises a number of questions concerning hypermedia systems - with particular reference to the Notecards system, and suggests how they should perhaps be extended in future versions or generations. These problems include providing search-and-query facilities, where navigational access alone may be insufficient; the use of composites to augment the basic model; the use of virtual structures to support information reorganisation; computation over networks and distributed collaboration; versioning; and extendability and configurability.

The architecture used in constructing GENIE-M systems, and in particular the geometric nature of tiles, provides possible solutions to a number of these problems. For example, concerning composition[76], a GENIE-M mechanism allows a given node to be included in more than one composite. The use of geometry to portray relationships between objects, as defined within GENIE-M systems, allows any node (tile) to be perceived (geometrically) either as part of a given composite (IDEA) or by the concept of the group. Versions of nodes can be introduced by placing more recent tiles geometrically above (or relatively 'before' or in advance of) older versions of the tile, and as stated previously, GENIE-M does not explicitly enable destruction of a tile, it merely provides for its relocation to a remote zone within the tile space.

Dynamic reorganisation of the tiles - and correspondingly the incorporation of changes to information - is supported in two ways: firstly, by using the modules which manipulate the tile space; and secondly by using the links which can exist explicitly (references) or implicitly (geometrically) between tiles. Inclusion of external systems for specific applications is provided using the concept of the external function supported within a complete environment.

Mechanisms for promoting consistent and coherent movement of data between multiple cooperating processes have been investigated and implemented. The same modular architecture which supports these processes is also conducive towards

76: Composition in this context is a way of dealing with groups of objects without necessarily having to consider their components.

upgrading and improvement by a simple and demonstrable mechanism for module replacement. The multi-tasking nature of GENIE-M has been referred to in section 7.7 as an example of an application technology suitable for employing advances in parallel computing hardware. Similarly, the use of higher level modules suggests the possibility of simulation tools with independent entities that perhaps may not be so cooperative, while an overseeing module can monitor the activities that ensue.

Having spent much time discussing the system specification, two implementation development areas were clearly defined. The emphasis of the present research work has been upon the development of a theoretical basis on which to design system internals, essential for constructing a prototype system. The goal of developing a true computer-based multimedia information environment generator naturally suggests the second major development area, concerning the construction of a very highly developed graphical front end. However, for the research under discussion at present, such a front end is a separate entity that requires the focus of an alternative research project. The primarily text-orientated mechanisms used for controlling the underlying multimedia manipulations are sufficient for the prototype. The text-based commands are designed to be used by application modules, of which an important example would be the graphical front-end.

As stated in section 10.2, the architecture is not suitable for construction or configuration by inexperienced users. Generating application modules is not necessarily as simple as fitting together basic components - although the architecture would eventually be developed to the point where such an approach would be mandatory. Instead, the present emphasis should be on the construction of applications within the higher-level of the information environment. Failing to incorporate this philosophy into the development of applications would be to undermine the conceptual advantages provided by the architecture.

The development of the architecture has been guided throughout by the desire to implement a technology for supporting information environments, and this in turn has meant that much of the work has had to be constructed from first principles. Use of existing tools may have unintentionally directed the skeleton of the project away from the generalised goal. For each instance of a problem, its nature has been categorised prior to an appropriate mechanism being generated. The mechanisms have included control tools, or device drivers, or higher level concepts such as presentation models.

The development experiments carried out have been both interesting and extremely instructive. The author's knowledge and experience has benefitted enormously in the many areas studied. The internal mechanisms developed to facilitate the functioning of GENIE-M have been seen to work, and have enabled the generation of a demonstration system which may already be of practical value.

The prototype system is helpful in identifying how the geometric mechanism can be used in analysing certain problems, and in the construction of the information environments to be used for contributing to tasks. In particular, the prototype has indicated areas where further research must be carried out. Even without the goal of generating information environments, the architecture suggests a number of mechanisms for implementing simpler applications, and in particular those involving multiple, concurrent processes.

Section 10.5: Limitations of the Current System: Threads for Future Development.

The prototype system represents a form of 'enabling technology'. The study of its construction and development is of inestimable value in providing pointers and guidance for future work. Limitations of the prototype focus more upon its included capabilities rather than faults in the basic concepts, and in this sense the prototype remains viable. However, in practical terms there are many ways in which the prototype could be extended into a more comprehensive 'production' system.

For example, the current form of the GENIE-M generator does not provide for the automatic construction of information systems. Instead it provides a series of tools (such that the word 'generate' is taken to mean provision of service) which can be used in the construction (generation) of a computer-based information system. Consequently, the generation has to be performed by an operator familiar with the system. However, this has the advantage that specific applications can be constructed and tailored to individual requirements, while maintaining the underlying system concepts.

The communication mechanisms are fully functional, but have room for improvement. In particular, the problem of multiple asynchronous message passing needs to be examined in more detail. This is partly to identify any machine- or operating-system dependencies or difficulties with using UNIX-based inter-process communications; and partly to improve the overall speed of module inter-communication. Messages can already pass between several modules concurrently, but the prototype restricts this so that the flow of messages through the system kernel can be controlled to avoid deadlock. As research into 'deadly embrace' problems progresses, so the results can be incorporated into the prototype. Other extensions to the message passing mechanism could incorporate a priority category, so that imperative messages such as 'imminent system failure' could be dealt with promptly.

The nature of the interconnection assumes reliable connections. A transaction log has been implemented to record all exchanges, but this is only of interest at present for debugging purposes. A useful enhancement to the system would incorporate a restart mechanism which would reconstruct the exact state of the application to the precise moment when any failure occurred.

Although UNIX provides the host operating system, and the majority of the prototype implementation was written using 'C', the design has been such that there are few restrictions on the choice of language used to implement similar versions, or to enhance the existing system. Those restrictions which do exist relate to the multi-tasking nature of the entire concept, and in particular to the need for suitable inter-module communications. The provision of fundamental primitives is intended to minimise the constraints that would otherwise be imposed by host machine, operating system or language. Deliberately changing machines to study implementation dependencies would be a very interesting task, but the experiences gained in implementing the demonstration system suggest that the implementation areas most likely to cause difficulties during any porting process have been clearly identified.

A clear objective in the future development of GENIE-M is the construction of a primarily graphics-orientated front-end. In much the same way that certain Graphic User Interfaces (GUI's) 'sit upon' otherwise complex operating system control mechanisms, so a graphic front-end for GENIE-M should provide much better accommodation of individual user requirements. This is because the graphic representation would be able to associate more closely with the underlying geometry of the application. Therefore, the key distinction between most GUI technology implementations and a GENIE-M graphic front-end would be the recognition of the geometric architecture which forms the basis of GENIE-M.

The GENIE-M work has suggested a mechanism for solving some of the problems of current strategies for developing computer-based information system. In particular, it suggests a direction to be taken in providing for multiple users wishing to perform multiple tasks across a distributed and potentially incompatible architecture. The work does not in any way represent a complete solution to this problem, but by providing a consistent control mechanism based upon a new approach to representing the problem domain, GENIE-M provides a way forward that is generalised and extendable.

A major issue to be addressed is the construction of a formal methodology which would be used to identifying those problems suited to representation using geometry, followed by derivation of the geometrical nature of the solution. Having been so closely involved with the construction of the internal architecture, the author is sometimes able to identify appropriate mechanisms on the basis of experience; but this is clearly lacking in precision. The existence of the GENIE-M prototype system would enable future research to proceed on the basis that a working system can be used as an experimental testbed to validate the formal methods.

Section 10.6: Final Thoughts.

There are almost no fields of computer science and computer-based information system construction that are not in some way associated with the work carried out during this research - at times it seemed that the work would draw upon all fields of expertise throughout computer science and computer-based information system development. The implications of some of the resulting work perhaps suggest that a more detailed exposition and study may be of value, possibly to investigate whether a unification mechanism is a viable prospect for future work.

In hindsight, the scope of this work has increased dramatically since its original conceptions as a mechanism for providing multiple editions of electronic books. Behind all of the work was a desire to seek consistency and applicability. The primary objective has been to identify a core which is suitable for use as the basis of an information architecture, that is expandable and widely applicable. In doing so, the specification of the core was guided by the preference for a minimal system, without a large number of procedures. A workable core could be extended in any desirable manner by use of appropriate procedures which would themselves be suggested by the nature of the core itself. Consequently, any concepts suggested during the work were explored in depth and at length, simply to ensure that limitations or restrictions would be minimal and that the core objectives would remain consistent.

The prototype system has been highly experimental in nature, and was never intended to be a production system. It forms an experimental test-bed on which ideas can be evaluated and examined, and so represents an 'enabling technology'. The focus has been directed towards recognition, definition, implementation and testing of totally new principles. The representation of the information environment is a complex task, that starts with the construction of device drivers to support the first approximation to the problem. All the drivers have had to be constructed with the requirement for underlying consistency always in mind. Construction of a production system is straightforward given the extensible nature of GENIE-M, but would require the effort of a complete research team for a long term enterprise. Consequently, the prototype should be viewed as a paradigm to be used as the starting point in the development of the more advanced system.

Appendix.

References.

[Aceves 85] J. J. Garcia Luna Aceves & A. A. Poggio, 'Computer-Based Multimedia Communications', COMPUTER, October 1985.

[Aceves 85b] J. J. Garcia Luna Aceves, 'Towards Computer-Based Multimedia Information Systems', Computer Message Systems 85, ed. R. Uhlig, pp. 61-77, 1985.

[Ackoff 67] R. L. Ackoff, 'Management Misinformation Systems', Management Science, Vol. 14, No. 4, December 1967, pp. B147-B156.

[Ahlsen 84] M. Ahlsen, A. Bjornerstedt, S. Britts, C. Hulten & L. Soderlund, 'An Architecture for Object Management in OIS', ACM Transactions on Office Information Systems, Vol. 2, No. 3, July 1984.

[Akass 89] C. Akass, News article on Software Development delays, Datalink, September 1989.

[Akscyn 84] R. Akscyn & D. L. McCracken, 'The ZOG Approach to Database Management', Proc. Trends and Applications Conf: Making Database Work, May 1984.

[Akscyn 87] R. M. Akscyn, D. L. McCracken & E. A. Yoder, 'KMS: A Distributed Hypertext System for Sharing Knowledge in Organisations', Hypertext 87 Papers, November 13-15, 1987.

[Akscyn 88] R. M. Akscyn, D. L. McCracken & E. A. Yoder, 'KMS: A Distributed Hypermedia System for Managing Knowledge in Organisations', Comms. ACM, Vol. 31, No. 7, July 1988, pp. 820-835.

[Allen 81] T. Allen, R. Nix & A. Perlis, 'PEN: A Hierarchical Document Editor', Proc. ACM SIGPLAN/SIGOA Symposium on Text Manipulation, Oregon, Vol. 2, Nos. 1 & 2, June 1981.

[Angell 86] I. O. Angell, Y. P. Low & A. R. Warman, 'GEODOC: An Overview', Internal Note 1906, Department of Computer Science, University of London, 19 February 1986.

[Angell 87] I. O. Angell, Y. P. Low & A. R. Warman, 'GENIE-M: A Generator for Multimedia Information Environments', British Computer Society Conference: 'Workstations and Publication Systems', London, 22-23 October 1986, papers published by Springer-Verlag 1987.

[Applix 84] Applix Inc., 'Alis - A next-generation office software system from Applix: Application Summary', 1984.

[Avison 85] D. E. Avison, 'Information Systems Development: A Data Base Approach', Blackwell Scientific Publications, 1985.

[Avison 88] D. E. Avison & G. Fitzgerald, 'Information Systems Development: Methodologies, Techniques and Tools', Blackwell Scientific Publications, 1988.

[Ayers 84] R. M. Ayers, J. T. Horning, & B. W. Lampson, 'Interscript: A Proposal for a Standard for the Interchange of Editable Documents', Xerox Palo Alto Research Center, California, 1984.

[Bach 86] M. J. Bach, 'The Design of the UNIX Operating System', Prentice-Hall, 1986.

[Backer 82] D. Backer & S. Gano, 'Dynamically Alterable Videodisk Displays', Proc. Graphics Interface 82, May 1982.

[Baker 72] F. T. Baker, 'Chief programmer team management of production programming', IBM Systems Journal, Vol. 11, No. 1

[Barber 83] G. R. Barber, 'Supporting Organisational Problem Solving with a Workstation', ACM Trans. on Office Information Systems, Vol. 1, No. 1, January 1983, pp. 45-67

[Baskin 80] A. B. Baskin, 'Logic Nets: Variable-valued Logic plus Semantic Networks', International Journal on Policy Analysis and Information Systems, Vol. 4, No. 269, 1980.

[Beeman 87] W. O. Beeman, K. T. Anderson, G. Bader, J. Larkin, A. P. McClard, P. J. McQuillan & M. Shields, 'Hypertext and Pluralism: From Lineal to Non-Lineal Thinking', Hypertext 87 Papers, November 13-15, 1987.

[Begeman 86] M. L. Begeman, P. Cook, C. Ellis, M. Graf, G. Rein & T. Smith, 'PROJECT NICK: Meetings Augmentation and Analysis', Computer-Supported Cooperative Work 86 Proceedings, December 3-5, 1986.

[Bender 84] W. Bender, 'Imaging and Interactivity', Fifteenth Joint Conf. on Image Technology, November 1984.

[Benyon 87] D. Benyon & S. Skidmore, 'Towards a Tool Kit for the Systems Analyst', The Computer Journal, Vol. 30, No. 1, February 1987.

[Bersoff 81] E. H. Bersoff, V. D. Henderson & S. G. Siegel, 'Software Configuration Management', Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[Bigelow 87] J. Bigelow & V. Riley, 'Manipulating Source Code in Dynamic Design', Hypertext 87 Papers, November 13-15, 1987.

[Birrel 80] A. D. Birrell & R. M. Needham, 'A Universal File Server', IEEE Trans. Soft. Eng., Vol. SE-6, No. 5, September 1980, pp. 450-453.

[Blank 83] J. Blank & M. J. Krijger, 'Software Engineering: Methods and Techniques', Wiley Interscience, 1983.

[Boehm 75] B. W. Boehm, 'The High Cost of Software', Practical Strategies for Developing Large Software Systems, ed. E. Horowitz, Addison-Wesley, 1975.

[Bolt 80] R. A. Bolt, 'Put-that-there: Voice and Gesture at the Graphics Interface', Computer Graphics, Vol. 15, No. 3, pp. 262-270, August 1980.

[Bolter 87] J. D. Bolter, 'Hypertext and Creative Writing', Hypertext 87 Papers, November 13-15, 1987.

[Bono 85] P. R. Bono, 'A Survey of Graphics Standards and their Role in Information Interchange', IEEE Computer, October 1985, pp. 63-75.

[Bourne 83] S. R. Bourne, 'The UNIX System', Addison-Wesley, 1983.

[Boyle 85] C. D. B. Boyle & M. R. B. Clarke, 'An Intelligent Mail Filter', People and Computers: Designing the Interface, Proc. BCS HCI Conf. 17-20 September 1985, pp. 331-341.

[Brown 84] M. H. Brown & R. Sedgewick, 'A System for Algorithm Animation', Computer Graphics, Vol. 18, No. 3, July 1984.

[Brown 86] P. J. Brown, 'Interactive Documentation', Software: Practice and Experience, pp. 291-299, March 1986.

[Brown 86b] P. J. Brown, 'Viewing Documents on a Screen', from 'CD-ROM: The New Papyrus', Lambert and Ropiequet (Eds.), Microsoft Press, 1986.

[Brown 87] P. J. Brown, 'Turning Ideas into Products: the Guide System', Hypertext 87 Papers, November 13-15, 1987.

[Brown 89] H. Brown, 'Standards for Structured Documents', Computer Journal, Vol. 32, No. 6, December 1989, pp. 505-514.

[Buckle 82] J. K. Buckle, 'Software Configuration Management', Macmillan, London, 1982.

[Bush 45] Vannevar Bush, 'As We May Think', Atlantic Monthly No. 176, pg 101-108, July 1945.

[Bush 67] Vannevar Bush, 'Memex Revisited', Science is Not Enough (Vannevar Bush), pp. 75-101, 1967.

[Butler 88] R. Butler, 'GIS - Geographic Information Systems - An Introduction', Mapping Awareness, Vol. 2, No. 2, May 1988, pp. 31-34.

[Byrd 82] R. J. Byrd, S. E. Smith & S. P de Jong, 'An Actor-Based Programming System', ACM SIGOA Conference on Office Information System, Vol. 3, Nos. 1 & 2, 1982.

[Campbell 87] B. Campbell & J. M. Goodman, 'HAM: A General Purpose Hypertext Abstract Machine', Hypertext 87 Papers, November 13-15, 1987.

[Campbell 88] J. Campbell, 'The C Programmer's Guide to Serial Communications', Howard Sams & Company, 1988.

[Cashin 73] P. Cashin, M. Robinson & D. Yates, 'Experience with SCRAPBOOK, a Non-Formatted Data Base System', Proc. IFIPS Congress, 1973.

[Chamberlin 82] D. D. Chamberlin, J. C. King, D. R. Slutz, S. J. P. Todd & B. W. Wade, 'JANUS: An Interactive Document Formatter based on Declarative Tags', IBM Systems Journal, Vol. 21, No. 3, 1982, pp. 250-271.

[Charney 87] D. Charney, 'Comprehending Non-Linear Text: The Role of Discourse Cues and Reading Strategies', Hypertext 87 Papers, November 13-15, 1987.

[Christodoulakis 86] S. Christodoulakis, M. Theodoridou, F. Ho, M. Papa & A. Pathira, 'Multimedia Document Presentation, Information Extraction, and Document Formation in MINOS: A Model and a System', ACM Trans. Office Information Systems, Vol. 4, No. 4, October 1986, pp. 345-383.

[Clark 83] P. A. Clark, BYTE Magazine, May 1983, McGraw-Hill Inc.

[Codd 70] E. F. Codd, 'A Relational Model of Data for Large Shared Data Banks', Communications of the ACM, June 1970.

[Coleman 88] D. Coleman & R. Gallimore, 'A Framework for Program Development', Computing Techniques, March 1988, pp 36-40.

[Collier 87] G. H. Collier, 'Thoth-II: Hypertext with Explicit Semantics', Hypertext 87 Papers, November 13-15, 1987.

[Conklin 87] J. Conklin, 'Hypertext: An Introduction and Survey', IEEE Computer, September 1987, pp. 17-41.

[Constantine 79] L. L. Constantine & E. Yourdon, 'Structured Design', Prentice-Hall, Englewood Cliffs, New Jersey, 1979.

[Cook 80] R. P. Cook, '*MOD - A Language for Distributed Programming', IEEE Trans. Soft. Eng., Vol. SE-6, No. 6, November 1980, pp. 563-571.

[Corda 86] U. Corda & G. Facchetti, 'Concept Browser: A System for Interactive Creation of Dynamic Documentation', Text Processing and Document Manipulation: Proceedings of the International Conference, ed. J. C. van Vliet, Cambridge University Press, 1986.

[Cox 83] Brad J. Cox, 'The Object Oriented Pre-Compiler', SIGPLAN Notices, Vol. 18, No. 1, January 1983.

[Cox 84] Brad J. Cox, 'Message/Object Programming: An Evolutionary Change in Programming Technology', IEEE Software, January 1984.

[Croft 88] W. B. Croft & L. S. Lefkowitz, 'Using a Planner to Support Office Work', Proc. Conf. Office Information Systems, ACM, 1988.

[deMarco 78] T. deMarco, 'Structured Analysis and System Specification', Yourdon Press, 1978.

[Denning 82] P. Denning, 'Electronic Junk', Comms. ACM, Vol. 23, 1982, pp. 163-165.

[Diamond 85] Bolt Beranek & Newman Inc., 'The Diamond Multimedia Document System: User's Reference Guide', Technical Report 6006, November 1985.

[di Sessa 85] A. di Sessa, 'A Principled Design for an Integrated Computational Environment', Human-Computer Interaction, Vol. 1, No. 1, 1985, pp. 1-47.

[di Sessa 86] A. di Sessa & H. Abelson, 'Boxer: A Reconstructable Computational Medium', Communications of the ACM, Vol. 29, No. 9, September 1986, pp. 859-868.

[Dijkstra 68] E. W. Dijkstra, 'Cooperating Sequential Processes', Programming Languages, Academic Press, New York, 1968.

[Downs 88] E. Downs, P. Clare & I. Coe, 'Structured Systems Analysis and Design Method: Application and Context', Prentice Hall International, 1988.

[DTI 87] Department of Trade and Industry, 'Software Tools for Application to Large Real Time Systems (STARTS)', H.M.S.O, 1987.

[ECMA 85] European Computer Manufacturers Association, 'Office Document Architecture', Standard ECMA-101, September 1985.

[Ehardt 83] J. L. Ehardt, 'Apple's Lisa: A Personal Office System', The Seybold Report on Office Systems, Vol. 6, No. 2, 24 January 1983.

[Ein-Dor 86] P. Ein-Dor & E. Segev, 'Attitudes, Association and Success of MIS: Some Empirical results from Research in the Context of a Business Game', Computer Journal, Vol. 29, No. 3, June 1986, pp. 212-221.

[Enderle 84a] G. Enderle, 'The Interface of the UIMS to the Application (Working Group Report)', Computer Graphics Forum 3, North Holland, 1984.

[Enderle 84b] G. Enderle, 'Seeheim Workshop on User Interface Management Systems (First Report)', Computer Graphics Forum 3, North Holland 1984.

[Engelbart 68] D. C. Engelbart & W. K. English, 'A Research Center for Augmenting Human Intellect', Proc. FJCC, Vol. 33, No. 1, December 9-11, 1968.

[Engelbart 78] D. C. Engelbart, 'Toward Integrated Evolutionary Office Automation Systems', Proceedings of the International Engineering Management Conference, October 16-18, 1978.

[Englebart 87] D. C. Engelbart, 'Hypertext 1987: Conference Proceedings', Chapel Hill, N.C., November 1987.

[Engelbart 88] D. C. Engelbart & H. Lehtman, 'Working Together', Byte Magazine, Vol. 13, No. 13, pp. 245-252, December 1988.

[Feiner 81] S. Feiner, S. Nagy & A. van Dam, 'An Integrated System for Creating and Presenting Complex Computer-Based Documents', Computer Graphics, Vol. 15, No. 3, August 1981, pp. 181-189.

[Feiner 82] S. Feiner, S. Nagy & A. Van Dam, 'An Experimental System for Creating and Presenting Interactive Graphical Documents', ACM Transactions of Graphics, Vol. 1, No. 1, January 1982, pp. 59-77.

[Fitzgerald 85] G. Fitzgerald, N. Stokes & J. R. G. Wood, 'Feature Analysis of Contemporary Information System Methodologies', The Computer Journal, Vol. 28, No. 3, 1985, pp. 223-230.

[Forsdick 82] H. C. Forsdick & R. H. Thomas, 'The Design of Diamond: A Distributed Multimedia Document System', Technical Report 5402, Bolt Beranek & Newman Inc., October 1982.

[Forsdick 85] H. C. Forsdick, 'Explorations into Real-Time Multimedia Conferencing', Proceeding of the Second International Symposium on Computer Messaging Systems, Sept. 1985.

[Foster 85] E. Foster, 'Outliners: A New Way of Thinking', Personal Computing, May 1985, pg. 74.

[Furuta 82a] R. Furuta, J. Scofield & S. Shaw, 'Document Formatting Systems: Survey, Concepts, and Issues', Computing Surveys, Vol. 14, No. 3, September 1982.

[Furuta 82b] R. Furuta, J. Scofield, & A. Shaw, 'Document Formatting Systems: Survey, Concepts and Issues: Document Preparation Systems', J. Nievergelt, G. Coray, J.D. Nicoud and A.C. Shaw (Eds.), North-Holland, 1982.

[Furuta 89] R. Furuta, 'An Object-based Taxonomy for Abstract Structure in Document Models', Computer Journal, Vol. 32, No. 6, December 1989, pp. 494-504.

[Gaines 86] B. R. Gaines & M. L. G. Shaw, 'Foundations of Dialog Engineering: The Development of Human-Computer Interaction', Int. J. Man-Machine Studies, Vol. 24, 1986, pp. 101-123.

[Galliers 87] R. D. Galliers & F. F. Land, 'Choosing Appropriate Information Systems Research Methodologies', Comms. ACM, Vol. 30, No. 11, pp. 900-902.

[Gane 79] C. Gane & T. Sarson, 'Structured Systems Analysis', Prentice-Hall, 1979.

[Gibbs 87] S. Gibbs, D. Tsichritzis, A. Fitas, D. Konstantas & Y. Yeorgaroudakis, 'Muse: A Multimedia Filing System', IEEE SOFTWARE, March 1987, pp. 4-15.

[Gibson 89] M. L. Gibson, 'The CASE Philosophy', Byte, April 1989, pp. 209-218.

[Goldberg 83] A. Goldberg & D. Robson, 'Smalltalk-80: The Language and its Implementation', Reading, MA: Addison-Wesley, 1983.

[Gomaa 83] H. Gomaa, 'The Impact of Rapid Prototyping on Specifying User Requirements', ACM Software Engineering Notes, Vol. 8, No. 2, pp. 17-28, 1983.

[Grindley 89] The Price Waterhouse 1989/90 Information Technology Survey, ed. C. P. P. Grindley.

[Gutknecht 84] J. Gutknecht & W. Winiger, 'Andra: The Document Preparation System of the Personal Workstation Lilith', Software - Practice and Experience, Vol. 14, pp. 73-100 1984.

[Gutknecht 85] J. Gutknecht, 'Concepts of the Text Editor Lara', Communications of the ACM, Vol. 28, No. 9, September 1985.

[Hakiel 87] S. R. Hakiel, 'Issues in the Design of Icons', Internal Research Paper, PLessey Telecommunications, 1987.

[Halasz 88] F. G. Halasz, 'Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems', Comms. ACM, Vol. 31, No. 7, July 1988, pp. 836-852.

[Hall 87] N. S. Hall, S. Laflin, W. P. Dodd, 'Integration of Graphic with Text in an Electronic Journal', Workstations and Publication Systems, Springer-Verlag, 1987.

[Hansen 88] W. J. Hansen & C. Haas, 'Reading and Writing with Computers: A Framework for Explaining Differences in Performance', Comms. ACM, Vol. 31, No. 9, September 1988, pp. 1080-1089.

[Harke 87] U. Harke, M. Burger & D. Gall, 'Embedding Graphics into Documents by Using a Graphics Editor', Workstations and Publication Systems, Springer-Verlag, 1987.

[Harrison 84] B. Harrison, 'FRAMEWORK: An Introduction', Ashton-Tate Publications, 1984.

[Henderson 86] D. A. Henderson & S. K. Card, 'Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface', ACM Trans. Graphics, Vol. 5, No. 3, July 1986, pp. 211-243.

[Heppe 85] D. L. Heppe, W. H. Edmondson & R. Spence, 'Helping Both the Novice and Advanced User in Menu-Driven Information Retrieval Systems', People and Computers: Designing the Interface, Proc. BCS HCI Conf. 17-20 September 1985, pp. 92-101.

[Herot 80] C. F. Herot, 'Spatial Management of Data', ACM Transactions on Database Systems, Vol. 5, No.4, December 1980.

[Hershey 85] W. Hershey, 'Idea Processors', Byte, pp 337-350, June 1985.

[Hewitt 77] C. Hewitt, 'Viewing Control Structures as Patterns of Passing Messages', Artificial Intelligence, Vol. 8, 1977.

[Hoare 78] C. A. R. Hoare, 'Communicating Sequential Processes', Communications of the ACM, Vol. 21, No. 8, August 1978, pp. 666-677.

[Honeywell 87] J. Honeywell, 'Publication Systems at TODAY', Workstations and Publication Systems, Springer-Verlag, 1987.

[Ince 89] D. Ince, 'The Software Prototype', .EXE Magazine, Vol. 4, No. 4, September 1989.

[ISO 84] ISO, 'Information Processing - Text Preparation and Interchange - Text Structures - Part 2: Office Document Architecture', ISO/TC 97/SC 18 N 267, April 1984.

[ISO SGML] 'Information Processing - Text and Office Systems - Standard Generalised Markup Language (SGML)', ISO Draft International Standard, ISO/DIS 8879 TC/97.

[Jackson 75] M. A. Jackson, 'Principles of Program Design', Academic Press, London, 1975.

[Jackson 83] M. A. Jackson, 'System Development', Prentice-Hall, London, 1983.

[Jadrnicek 84] R. Jadrnicek, 'SYMPHONY: A Full-Orchestra Version of Lotus 1-2-3', BYTE, July 1984.

[Jones 89] R. Jones, 'Prototyping: Roughing it up', Datalink, 25 September 1989.

[Kiesler 89] S. Kiesler, 'Are Friends Electric: E-mail behaviour', PC Business World, October 1989.

[Kimura 86] G. D. Kimura, 'A Structure Editor for Abstract Document Objects', IEEE Transactions on Software Engineering, Vol. SE-12, No. 3, March 1986.

[Konsynski 85] B. R. Konsynski, A. Greenfield & W. E. Bracker Jr., 'A View on Windows: Current Approaches and Neglected Opportunities', AFIPS Conference Proceedings, 1985 National Computer Conference, Vol. 54, AFIPS Press 1985.

[Knuth 79] D. E. Knuth, 'T_EX and METAFONT: A New Direction in Typesetting', Digital Press, 1979.

[Knuth 84] D. E. Knuth, 'Literate Programming', The Computer Journal, Vol. 27, No. 2, 1984, pp. 97-111.

[Knuth 86] D. E. Knuth, 'The T_EXbook', Addison-Wesley Publishing, Reading, MA 1986.

[Lopez 73] B. Lopez, writing in 'Environment Action', 31 March, 1973.

[Machlup 83] F. Machlup & U. Mansfield, eds., 'The Study of Information', Wiley, 1983.

[Maddison 83] R. N. Maddison, 'Information System Methodologies', Wiley, London, 1983.

[Madnick 83] S. E. Madnick & J. J. Donovan, 'Operating Systems', McGraw-Hill, 9th edition, 1983.

[Malone 87] T. W. Malone, K. R. Grant, F. A. Turbak, S. A. Brobst & M. D. Cohen, 'Intelligent Information Sharing Systems', Comms. ACM, Vol. 30, 1987, pp. 390-402.

[MASCOT 83] 'The Official Handbook of MASCOT', RSRE Malvern, 1983.

[Maude 85] T. I. Maude & D. J. Pullinger, 'Software for Reading, Refereeing and Browsing in the BLEND System', Computer Journal, Vol. 28, No. 1, February 1985, pp. 1-4.

[Mayer 85] A. J. W. Mayer, 'Storage Architectures', Byte Magazine, December 1985, pp. 221-234.

[McCandless 90] H. McCandless, 'Mimic the mind's eye', Practical Computing, February 1990, pp. 38-42.

[McCracken 84] D. L. McCracken & R. M. Akscyn, 'Experience with the ZOG Human-Computer Interface System', International Journal of Man-Machine Studies, Vol. 21, 1984.

[Nanard 87] J. Nanard, M. Nanard & G. Cottin, 'Pleiade: A System for Interactive Manipulation of Structured Documents', Workstations and Publication Systems, Springer-Verlag, 1987.

[NCC 86] National Computing Centre, 'SSADM Manual', NCC Publications, 1986.

[Negroponte 76] N. Negroponte, 'An Idiosyncratic Systems Approach to Interactive Graphics', ACM/SIGGRAPH Workshop paper, Pittsburgh, Penn., October 14-15, 1976.

[Nelson 67] T. H. Nelson, 'Getting It Out of Our System', Information Retrieval: A Critical Review, editor G. Schechter, Thompson Books, 1967.

[Ngoh 89] L. H. Ngoh & T. P. Hopkins, 'Transport Protocol Requirements for Distributed Multimedia Information Systems', The Computer Journal, Vol. 32, No. 3, pp. 252-261, June 1989.

[Nicolis 77] G. Nicolis & I. Prigogine, 'Self-Organisation in Non-Equilibrium Systems: From Dissipative Structures to Order through Fluctuations', Wiley, 1977.

[Olsen 84] D. R. Olsen Jr., W. Buxton, R. Ehrich, D. J. Kasik, J. R. Rhyne & J. Sibert, 'A Context for User Interface Management', IEEE Computer Graphics and Application, December 1984.

[Oren 88] T. Oren, 'The CD-ROM Connection', BYTE Magazine, Vol. 13, No. 13, December 1988, pp. 315-320.

[Osin 76] L. Osin, 'SMITH: How to produce CAI Courses without Programming.' International Journal of Man-Machine Studies, Vol. 8, 1976.

[Ovum 87] Ovum, 'Computer Aided Software Engineering: Commercial Strategies', 1987.

[Panko 88] R. R. Panko, 'End User Computing: Management, Applications, and Technology', Wiley, 1988.

[PCMag 89] 'Of Mice and Menus', PC Magazine, October 1989, pp. 44-68.

[PCWeek 88] Report on Information Centre survey by Crwth Computer Coursewares, PC Week, 4 October 1988, p. 46.

[Pearce 89] A. Pearce, 'A Model of Analysis', Systems International, June 1989, pp. 71-74.

[Peels 85] A. J. H. M. Peels et. al., 'Document Architecture and Text Formatting', ACM Transactions on Office Information Systems, Vol. 3, No. 4, October 1985.

[Peters 80] L. J. Peters, 'Software Representation and Composition Techniques', Proceedings of the IEEE, Vol. 68, No. 9, August 1980, pp 1085-1093.

[Peterson 77] J. L. Peterson, 'Petri Nets', Computing Surveys, Vol. 9, No. 3, September 1977.

[Pitman 85] K. M. Pitman, 'CREF: An Editing Facility for Managing Structured Text', A.I. Memo No. 829, M.I.T. A.I. Laboratory, Cambridge, Mass., February 1985.

[Poggio 85] A. Poggio, J. J. Garcia Luna Aceves, E. J. Craighill, D. Moran, L. Aguilar, D. Worthington & J. Hight, 'CCWS: A Computer-Based Multimedia Information System', IEEE Computer, October 1985, pp. 92-103.

[Powell 89] M. Powell, 'Power Ahead with a 386', Computer Weekly, July 20, 1989, pg. 22.

[Ramo 69] S. Ramo, 'Cure for Chaos: Fresh Solutions to Social Problems Through the Systems Approach', New York, 1969.

[Randell 86] B. Randell, 'System Design and Structuring', Computer Journal, Vol. 29, No. 4, 1986, pp. 300-306.

[Reynolds 85] J. K. Reynolds, J. B. Postel, A. R. Katz, G. G. Finn & A. L. DeSchon, 'The DARPA Experimental Multimedia Mail System', IEEE Computer, October 1985, pp. 82-89.

[Rheingold 87] H. Rheingold, 'Tools for Thought', New York, 1987.

[Ritchie 74] D. M. Ritchie & K. Thompson, 'The UNIX Time-Sharing System', Communications of the ACM, Vol. 17, No. 7, July 1974, pp. 365-375.

[Ritchie 89] I. Ritchie, 'HYPERTEXT - Moving Towards Large Volumes', Computer Journal, Vol. 32, No. 6, December 1989, pp. 516-523.

[Roszak 86] T. Roszak, 'The Cult of Information', Lutterworth Press, 1986.

[Rummens 89] N. Rummens & R. Sucher, 'Competitive Edge', Systems International, March 1989.

[Sakata 85] S. Sakata & T. Ueda, 'A Distributed Interoffice Mail System', IEEE Computer, October 1985, pp. 106-116

[Sarin 85] S. Sarin & I. Greif, 'Computer-Based Real-Time Conferencing Systems', IEEE COMPUTER, October 1985, pp. 33-45.

[Sarin 87] S. K. Sarin & N. A. Lynch, 'Discarding Obsolete Information in a Replicated Database System', IEEE Trans. Software Engineering, Vol. SE-13, No. 1, January 1987, pp. 39-47.

[Scarrott 85] G. G. Scarrott, 'Information, the Life Blood of Organisations', Computer Journal, Vol. 28, No. 3, July 1985, pp. 203-205.

[Schifreen 88] R. Schifreen, 'Exchanging Data Dynamically', Personal Computer World, January 1988, pp. 158-162.

[Scott 86] J. Scott, 'Drivers Free Users from Device Dependence', Mini-Micro Systems, April 15, 1986, pp. 19-27.

[Shasha 85] D. Shasha, 'NetBook - a Data Model to Support Knowledge Exploration', Proceedings of VLDB 85, Stockholm, August 1985.

[Shasha 86] D. Shasha, 'When Does Non-Linear Text Help?', Proc. First International Conf. Expert Database Systems, pp. 109-121, April 1986.

[Shneiderman 87] B. Shneiderman, 'Designing the User Interface: Strategies for Effective Human-Computer Interaction', Addison-Wesley, 1987.

[Smith 82] D. C. Smith, C. Irby, R. Kimball & B. Verplank, 'Designing the Star User Interface', BYTE, April 1982.

[Smith 88] J. B. Smith & S. F. Weiss, 'Hypertext', Comms. ACM (Special Issue on Hypertext), Vol. 31, No. 7, July 1988, pp. 816-819.

[Sommerville 89] I. Sommerville, 'Software Engineering, 3rd Ed.', Addison-Wesley, 1989.

[Stamper 71] R. K. Stamper, 'Some Ways of Measuring Information', Computer Bulletin, December 1971, pp. 432-436.

[Stamper 85] R. K. Stamper, 'Information: Mystical Fluid or a Subject for Scientific Enquiry', Computer Journal, Vol. 28, No. 3, July 1985, pp. 195-199.

[Stobie 87] I. Stobie, 'The Integrated Office: When Technologies Converge', Practical Computing, September 1987.

[Straub 89] B. Straub & I. O. Angell, 'A Question of System', Working Paper Series No. 7, Department of Information Systems: London School of Economics, 1989.

[Tagg 87] R. Tagg, 'Software Engineering from the End User Angle', Computer Bulletin, December 1987, pp 12-13.

[Tazelaar 88] J. M. Tazelaar, 'Groupware in Depth', Byte Magazine, Vol. 13, No. 13, pg. 242, December 1988.

[Teichroew 77] D. Teichroew & E. A. Hershey, 'PSL/PSA: A computer-aided technique for structured documentation and analysis of information processing systems', IEEE Trans. Soft. Eng., Vol. SE-3, No. 1.

[Thoma 85] G. R. Thoma, S. Suthasinekul, F. L. Walker, J. Cookson & M. Rashidian, 'A Prototype System for the Electronic Storage and Retrieval of Document Images', ACM Trans. on Office Information Systems, Vol. 3, No. 3, July 1985, pp. 279-291.

[Thomas 83] J. J. Thomas (Workshop Chairman), 'Graphical Input Interaction Technique Workshop Summary', Computer Graphics, Vol. 17, No. 1, January 1983.

[Thomas 85] R. H. Thomas, H. C. Forsdick, T. R. Crowley, G. G. Robertson, R. W. Schaaf, R. S. Tomlinson & V. M. Travers, 'Diamond: A Multimedia Message System Built Upon a Distributed Architecture', COMPUTER, October 1985.

[Times 88] The Times, June 27th 1988. First report on Paris Airshow disaster.

[Times 88b] The Times, June 29th 1988. Preliminary report on Paris Airshow Disaster lays blame on pilot for ignoring 'too low' height warnings.

[Toffler 80] A. Toffler, 'The Third Wave', Morrow, 1980.

[Trigg 86] R. H. Trigg & M. Weiser, 'TEXTNET: A Network-Based Approach to Text Handling', ACM Trans. Office Information Systems, Vol. 4. No. 1, January 1986.

[UCL 83] University College London Department of Computer Science, 'UNIX Programmer's Manual, 4th Ed.', 1983.

[Walker 81] J. H. Walker, 'The Document Editor: A Support Environment for Preparing Technical Documents', Proc. ACM SIGPLAN/SIGOA Symposium on Text Manipulation, Oregon, Vol. 2, Nos. 1 & 2, June 1981.

[Walker 85] J. H. Walker, 'The Document Examiner', SIGGRAPH Video Review, CHI'85: Human Factors in Computing Systems, 1985.

[Ware 89] C. Ware, 'A C.A.S.E. in Point', Datalink, 25 September 1989.

[Warman 87] A. R. Warman, 'GENIE-M: Applications of Multimedia Information Environments', 'ED'87: Electronic Displays and Information Display Systems', London, 17-19 November 1987.

[Weiner 50] N. Weiner, 'The Human Use of Human Beings', Houghton Mifflin, 1950.

[Weyer 82] S. A. Weyer, 'The Design of a Dynamic Book for Information Search', International Journal of Man-Machine Studies, Vol. 17, 1982.

[Weyer 85] S. A. Weyer & A. H. Borning, 'A Prototype Electronic Encyclopedia', ACM Trans. on Office Information Systems, Vol. 3, No. 1, January 1985, pp. 63-88.

[White 87] E. White & R. Grehan, 'OS/2: Microsoft's New DOS', Byte Magazine, June 1987, pp. 116-126

[Whybrow 89] M. Whybrow, 'Substantiate the CASE Mirage', Infomatics: Software Tools, pp. 6-10, June 1989.

[Wilson 85] M. D. Wilson, P. J. Barnard & A. MacLean, 'Analysing the Learning of Command Sequences in a Menu System', People and Computers: Designing the Interface, Proc. BCS HCI Conf. 17-20 September 1985, pp. 63-75.

[Wilson 85b] P. A. Wilson, 'Mailbox Advances and MMI Needs', People and Computers: Designing the Interface, Proc. BCS HCI Conf. 17-20 September 1985, pp. 317-330.

[Winograd 87] T. Winograd, 'A Language/Action Perspective on the Design of Cooperative Work', Human-Computer Interaction, Vol. 3, No. 1, pp. 3-30, 1987.

[Winograd 88] T. Winograd, 'Where the Action Is', Byte Magazine, Vol. 13, No. 13, pp. 256-268, December 1988.

[Witten 85] I. H. Witten & B. Bramwell, 'A System for Interactive Viewing of Structured Documents', Comms. ACM, Vol. 28, No. 3, pp. 280-288

[Yankelovich 85] N. Yankelovich, N. Meyrowitz & A. van Dam, 'Reading and Writing the Electronic Book', IEEE COMPUTER, pp. 15-30, October 1985.

[Yankelovich 88] N. Yankelovich, B. J. Haan, N. Meyrowitz & S. M. Drucker, 'Intermedia: The Concept and the Construction of a Seamless Information Environment', IEEE Computer, Vol. 21, No. 1, January 1988, pp. 81-86.

[Yourdon 78] E. Yourdon & L. L. Constantine, 'Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design', New York: Yourdon Press, 1978.

Error Codes returned by the Communications Routines.

These codes are described in the form of UNIX-like error numbers, as UNIX is the host system used for constructing the prototype of the GENIE-M generator. The actual codes are declared as constants within the source for the GENIE-M implementation, and may be referenced by any program. The non-sequential values are a result of attempting to match the error codes to similar returns from standard UNIX system calls.

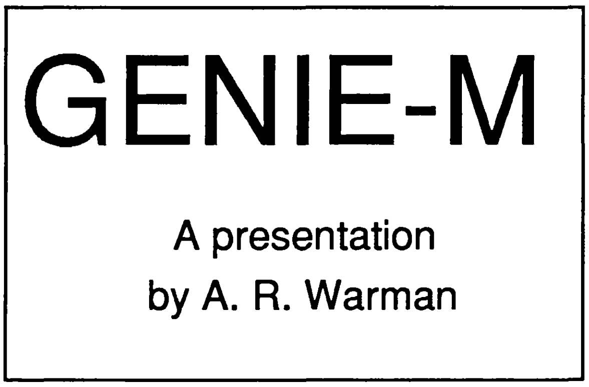
All GENIE-M functions and utilities will return a 0 error code on success, and a non-zero code if an error occurs. The non-zero code returned does not necessarily correspond with the actual error, instead the actual error code will be stored in the global variable `GENIE_errno`. This variable is not cleared on each successful call, and so should only be tested after an error has been detected.

0	GNOK	Error state 0, no error.
1	GNHOST	A host error has occurred. For the UNIX operating system, the error code will be located in the global access `errno` variable.
2	GNOENT	The requested entity does not exist. May refer to a tile, group, device or module.
5	GNIO	I/O error, typically the result of attempting to give an illegal instruction to a specific device, for example reading from a screen display.
7	GN2BIG	A parameter has been provided that is too long. This would occur if the destination of a `GENIE_send()` is not available in the current session, and is being referred to external hosts.

12	GNOMEM	Not enough memory. Caused by a variety of conditions - normally reflects a problem interacting with the host operating system.
13	GNACCESS	General permission denied.
14	GNFAULT	Faulty address or name. Could not identify the requested name or address.
16	GNBUSY	The requested resource is busy.
17	GNEXISTS	An entity of that name already exists.
22	GNINVAL	An invalid argument has been encountered in the message.
26	GNOBJBSY	The requested object (tile, group, device, module) is already in use.
32	GNSEND	The send operation has failed. This is probably due to a communications failure rather than a fault in locating the destination.
39	GNDEST	Could not locate the required destination anywhere in the application system.
58	GNSHUTDN	Cannot use this command, the system has been shutdown - or is not yet active.
60	GNTIMOUT	The requested operation timed out (could not be performed in a reasonable amount of time).
100	GNBADOP	Instruction would result in illegal state - for example tiles intersecting within the tile space.

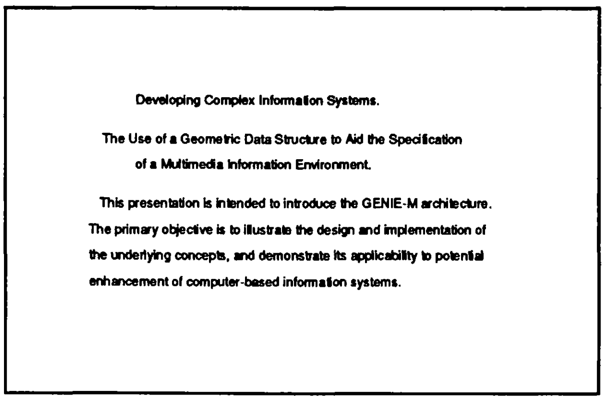
Script of Demonstration Session.

In keeping with the philosophy of the GENIE-M architecture, each instance of an electronic document would be tailored to meet the individual requirements of each reader. Such an objective is not possible with a purely paper-based document, but to reflect the principles, this section of the appendix includes a record of the script and diagrams used during the presentation of the demonstration system.



GENIE-M

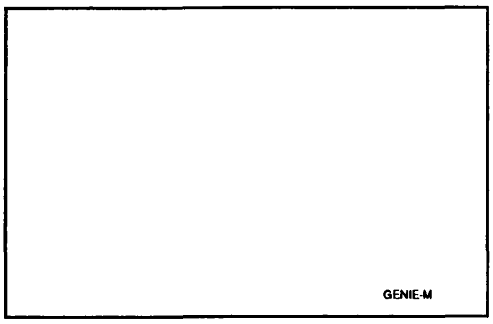
A presentation
by A. R. Warman



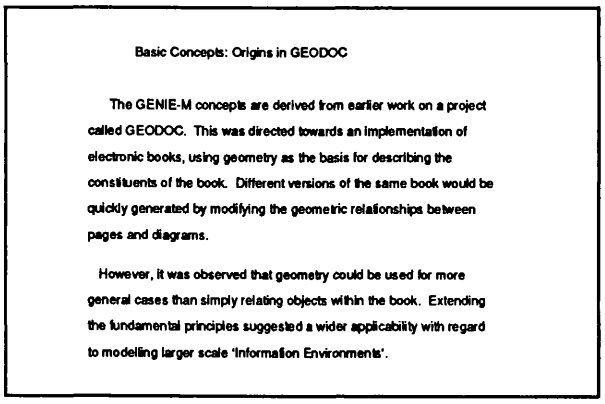
Developing Complex Information Systems.

The Use of a Geometric Data Structure to Aid the Specification of a Multimedia Information Environment.

This presentation is intended to introduce the GENIE-M architecture. The primary objective is to illustrate the design and implementation of the underlying concepts, and demonstrate its applicability to potential enhancement of computer-based information systems.



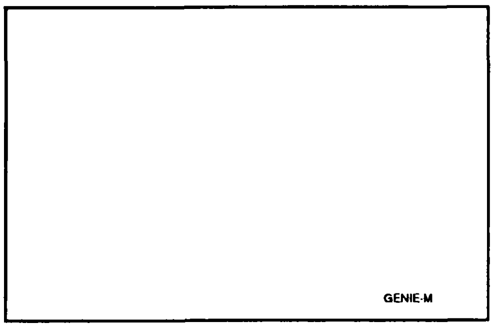
GENIE-M



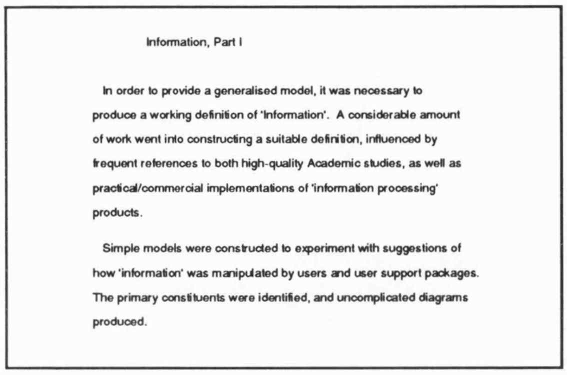
Basic Concepts: Origins in GEODOC

The GENIE-M concepts are derived from earlier work on a project called GEODOC. This was directed towards an implementation of electronic books, using geometry as the basis for describing the constituents of the book. Different versions of the same book would be quickly generated by modifying the geometric relationships between pages and diagrams.

However, it was observed that geometry could be used for more general cases than simply relating objects within the book. Extending the fundamental principles suggested a wider applicability with regard to modelling larger scale 'Information Environments'.



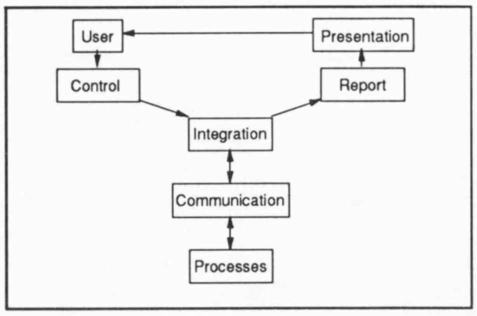
GENIE-M

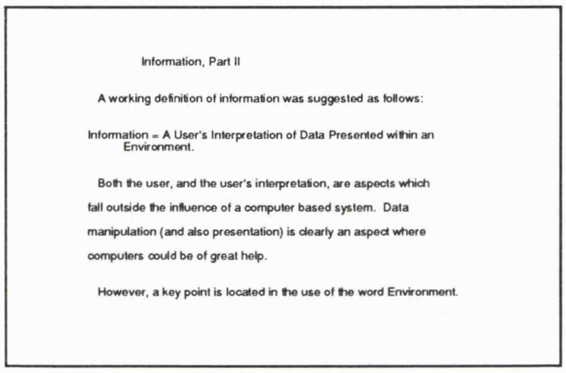


Information, Part I

In order to provide a generalised model, it was necessary to produce a working definition of 'Information'. A considerable amount of work went into constructing a suitable definition, influenced by frequent references to both high-quality Academic studies, as well as practical/commercial implementations of 'information processing' products.

Simple models were constructed to experiment with suggestions of how 'information' was manipulated by users and user support packages. The primary constituents were identified, and uncomplicated diagrams produced.





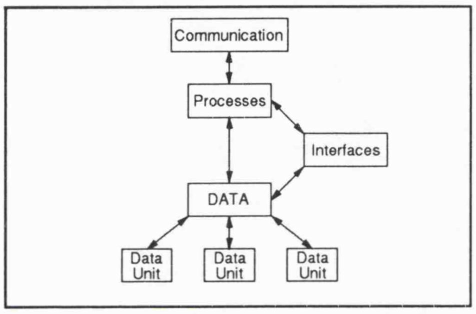
Information, Part II

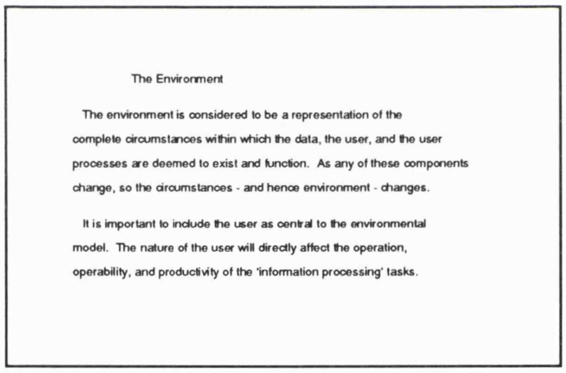
A working definition of information was suggested as follows:

Information = A User's Interpretation of Data Presented within an Environment.

Both the user, and the user's interpretation, are aspects which fall outside the influence of a computer based system. Data manipulation (and also presentation) is clearly an aspect where computers could be of great help.

However, a key point is located in the use of the word Environment.

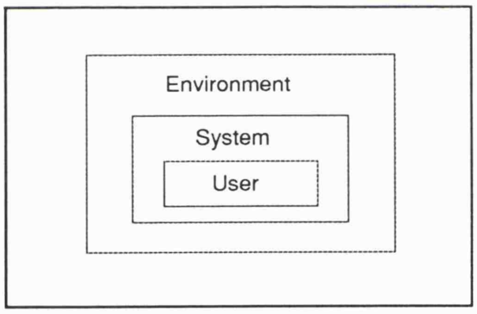


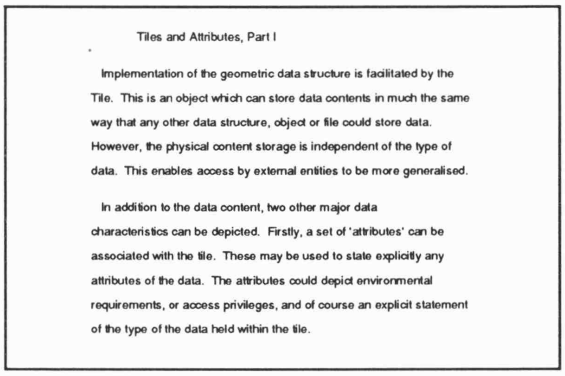


The Environment

The environment is considered to be a representation of the complete circumstances within which the data, the user, and the user processes are deemed to exist and function. As any of these components change, so the circumstances - and hence environment - changes.

It is important to include the user as central to the environmental model. The nature of the user will directly affect the operation, operability, and productivity of the 'information processing' tasks.

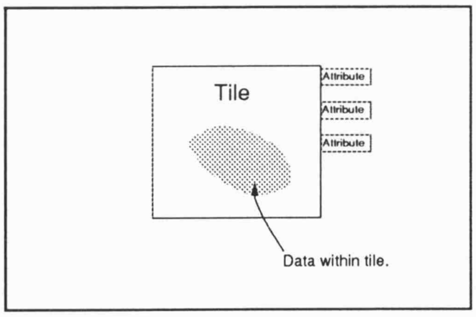


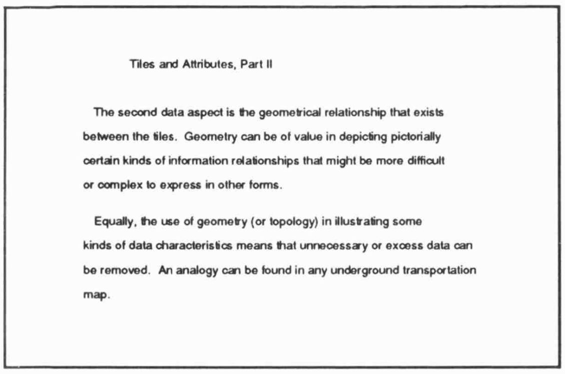


Tiles and Attributes, Part I

Implementation of the geometric data structure is facilitated by the Tile. This is an object which can store data contents in much the same way that any other data structure, object or file could store data. However, the physical content storage is independent of the type of data. This enables access by external entities to be more generalised.

In addition to the data content, two other major data characteristics can be depicted. Firstly, a set of 'attributes' can be associated with the tile. These may be used to state explicitly any attributes of the data. The attributes could depict environmental requirements, or access privileges, and of course an explicit statement of the type of the data held within the tile.

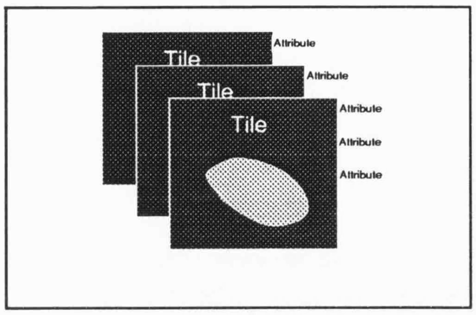


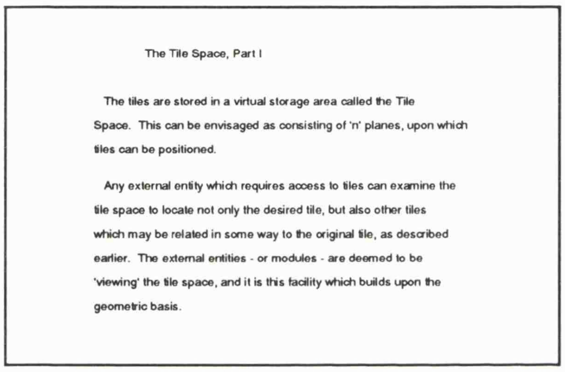


Tiles and Attributes, Part II

The second data aspect is the geometrical relationship that exists between the tiles. Geometry can be of value in depicting pictorially certain kinds of information relationships that might be more difficult or complex to express in other forms.

Equally, the use of geometry (or topology) in illustrating some kinds of data characteristics means that unnecessary or excess data can be removed. An analogy can be found in any underground transportation map.

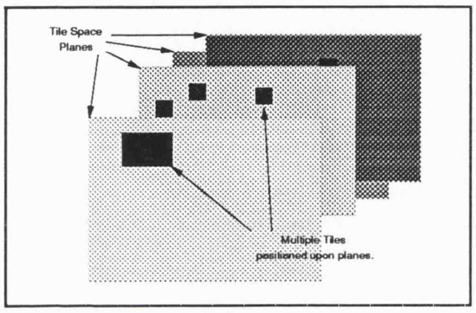


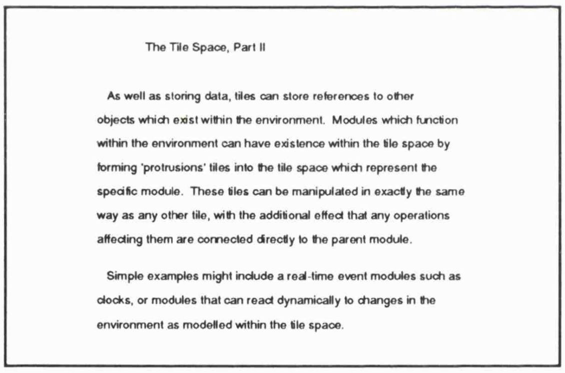


The Tile Space, Part I

The tiles are stored in a virtual storage area called the Tile Space. This can be envisaged as consisting of 'n' planes, upon which tiles can be positioned.

Any external entity which requires access to tiles can examine the tile space to locate not only the desired tile, but also other tiles which may be related in some way to the original tile, as described earlier. The external entities - or modules - are deemed to be 'viewing' the tile space, and it is this facility which builds upon the geometric basis.

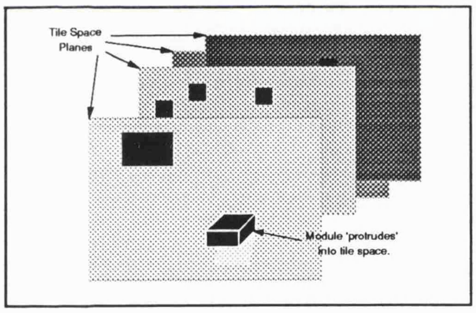


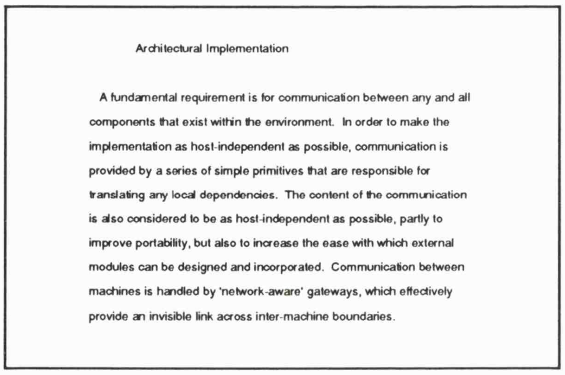


The Tile Space, Part II

As well as storing data, tiles can store references to other objects which exist within the environment. Modules which function within the environment can have existence within the tile space by forming 'protrusions' tiles into the tile space which represent the specific module. These tiles can be manipulated in exactly the same way as any other tile, with the additional effect that any operations affecting them are connected directly to the parent module.

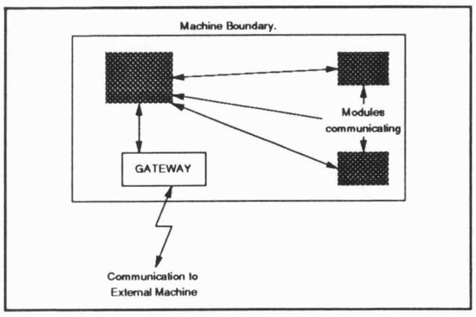
Simple examples might include a real-time event modules such as clocks, or modules that can react dynamically to changes in the environment as modelled within the tile space.

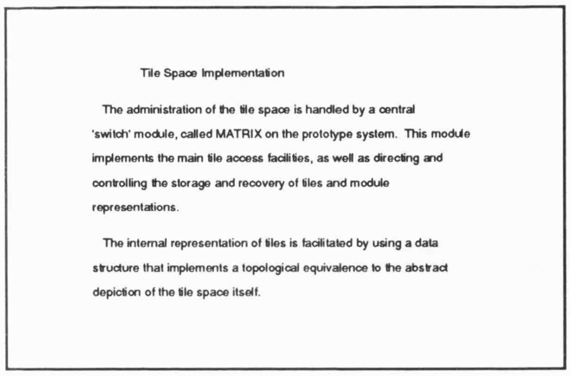




Architectural Implementation

A fundamental requirement is for communication between any and all components that exist within the environment. In order to make the implementation as host-independent as possible, communication is provided by a series of simple primitives that are responsible for translating any local dependencies. The content of the communication is also considered to be as host-independent as possible, partly to improve portability, but also to increase the ease with which external modules can be designed and incorporated. Communication between machines is handled by 'network-aware' gateways, which effectively provide an invisible link across inter-machine boundaries.

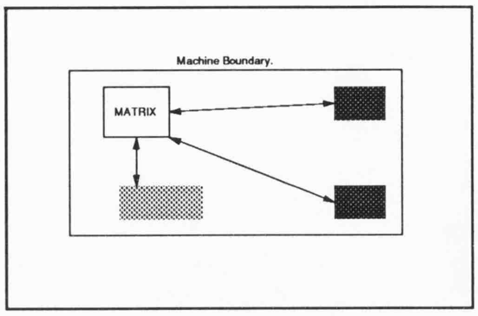


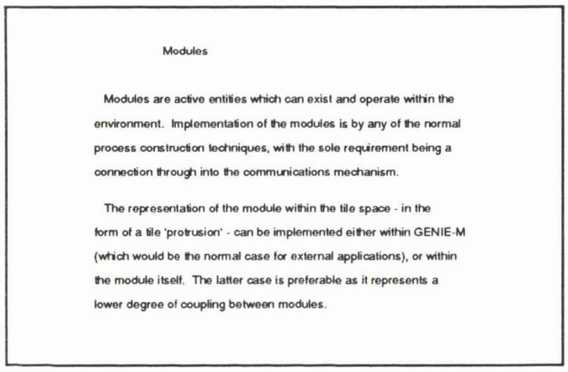


Tile Space Implementation

The administration of the tile space is handled by a central 'switch' module, called MATRIX on the prototype system. This module implements the main tile access facilities, as well as directing and controlling the storage and recovery of tiles and module representations.

The internal representation of tiles is facilitated by using a data structure that implements a topological equivalence to the abstract depiction of the tile space itself.

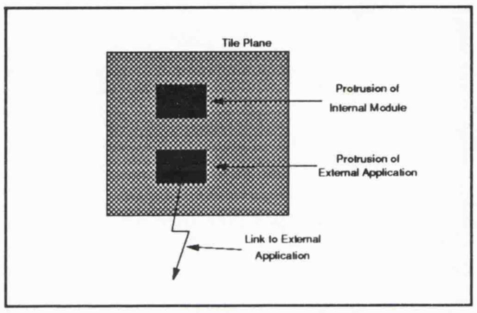


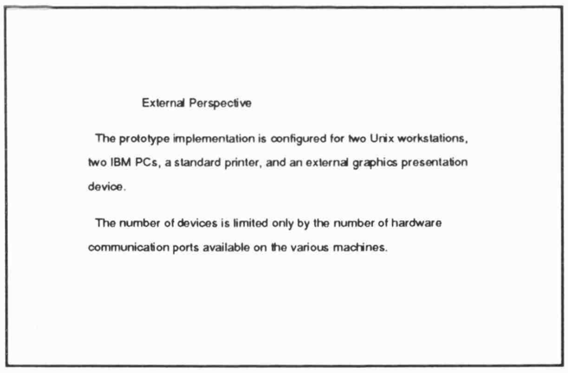


Modules

Modules are active entities which can exist and operate within the environment. Implementation of the modules is by any of the normal process construction techniques, with the sole requirement being a connection through into the communications mechanism.

The representation of the module within the tile space - in the form of a tile 'protrusion' - can be implemented either within GENIE-M (which would be the normal case for external applications), or within the module itself. The latter case is preferable as it represents a lower degree of coupling between modules.

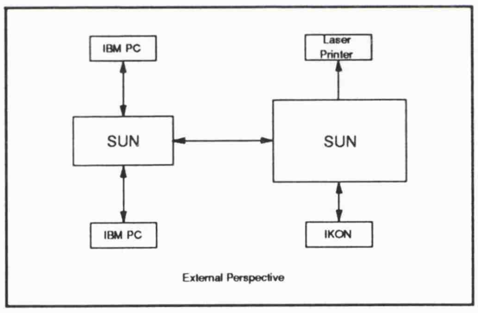

Tile Plane
Protrusion of Internal Module
Protrusion of External Application
Link to External Application

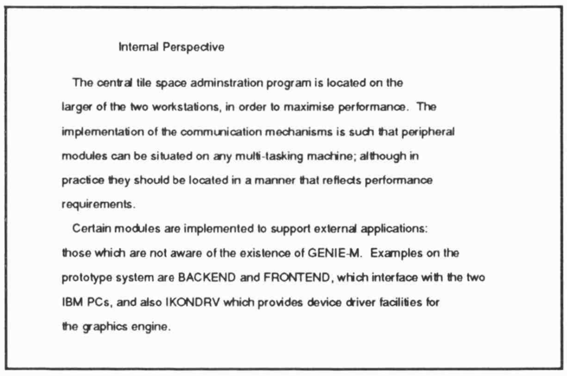


External Perspective

The prototype implementation is configured for two Unix workstations, two IBM PCs, a standard printer, and an external graphics presentation device.

The number of devices is limited only by the number of hardware communication ports available on the various machines.

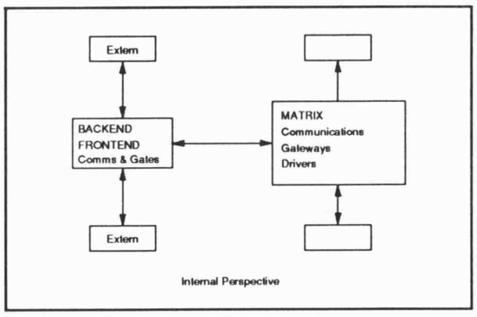


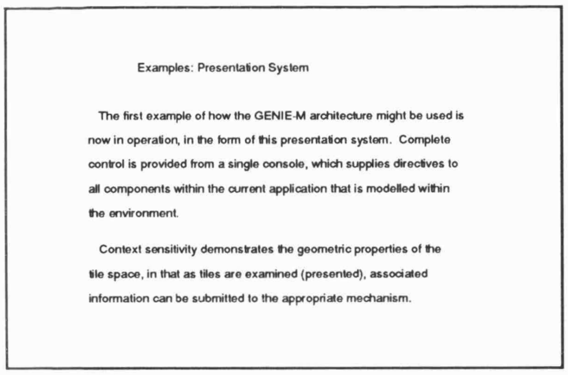


Internal Perspective

The central tile space adminstration program is located on the larger of the two workstations, in order to maximise performance. The implementation of the communication mechanisms is such that peripheral modules can be situated on any multi-tasking machine; although in practice they should be located in a manner that reflects performance requirements.

Certain modules are implemented to support external applications: those which are not aware of the existence of GENIE-M. Examples on the prototype system are BACKEND and FRONTEND, which interface with the two IBM PCs, and also IKONDRV which provides device driver facilities for the graphics engine.

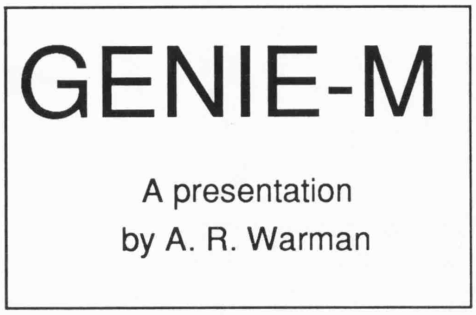

Extern
BACKEND FRONTEND Comms & Gates
MATRIX Communications Gateways Drivers
Extern
Internal Perspective



Examples: Presentation System

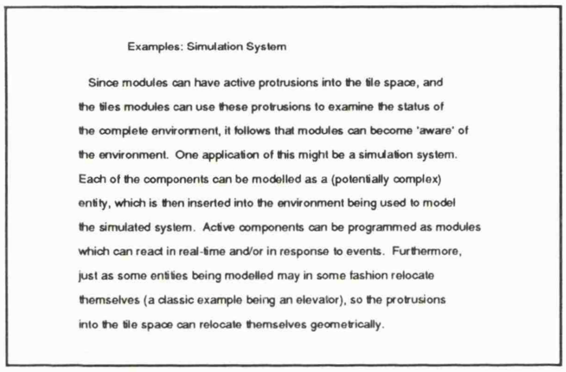
The first example of how the GENIE-M architecture might be used is now in operation, in the form of this presentation system. Complete control is provided from a single console, which supplies directives to all components within the current application that is modelled within the environment.

Context sensitivity demonstrates the geometric properties of the tile space, in that as tiles are examined (presented), associated information can be submitted to the appropriate mechanism.



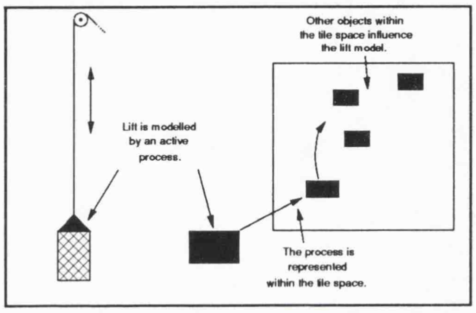
GENIE-M

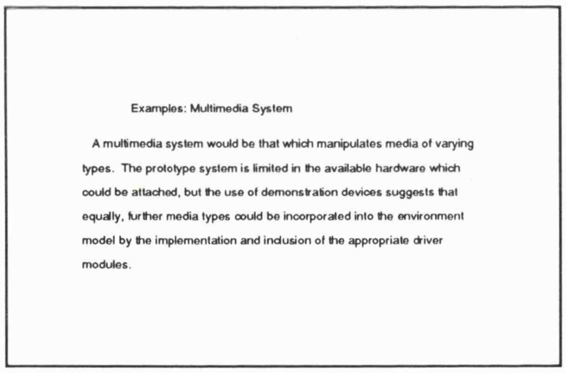
A presentation
by A. R. Warman



Examples: Simulation System

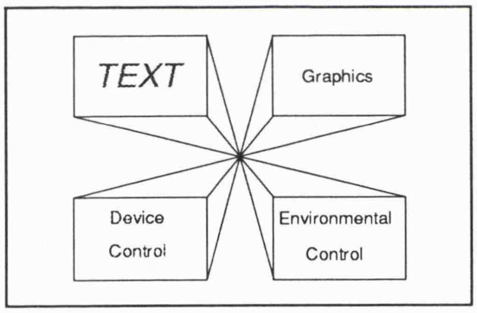
Since modules can have active protrusions into the tile space, and the tiles modules can use these protrusions to examine the status of the complete environment, it follows that modules can become 'aware' of the environment. One application of this might be a simulation system. Each of the components can be modelled as a (potentially complex) entity, which is then inserted into the environment being used to model the simulated system. Active components can be programmed as modules which can react in real-time and/or in response to events. Furthermore, just as some entities being modelled may in some fashion relocate themselves (a classic example being an elevator), so the protrusions into the tile space can relocate themselves geometrically.

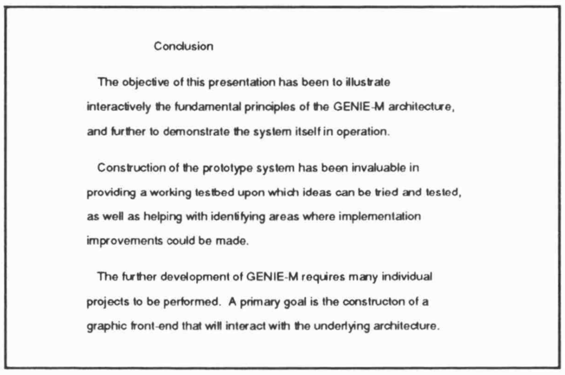

Other objects within the tile space influence the lift model.
Lift is modelled by an active process.
The process is represented within the tile space.



Examples: Multimedia System

A multimedia system would be that which manipulates media of varying types. The prototype system is limited in the available hardware which could be attached, but the use of demonstration devices suggests that equally, further media types could be incorporated into the environment model by the implementation and inclusion of the appropriate driver modules.



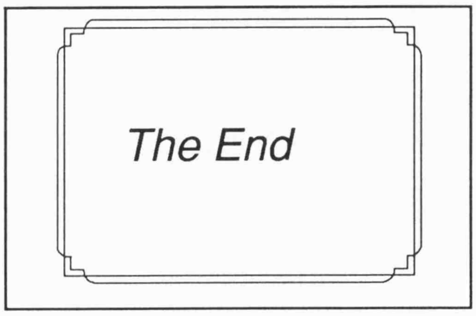


Conclusion

The objective of this presentation has been to illustrate interactively the fundamental principles of the GENIE-M architecture, and further to demonstrate the system itself in operation.

Construction of the prototype system has been invaluable in providing a working testbed upon which ideas can be tried and tested, as well as helping with identifying areas where implementation improvements could be made.

The further development of GENIE-M requires many individual projects to be performed. A primary goal is the construction of a graphic front-end that will interact with the underlying architecture.



The End