ISSN 0254–2757

QI QUÆSTIONES INFORMATICÆ

Volume 6 • Number 4 April 1989

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUÆSTIONES INFORMATICÆ

The official journal of the Computer Society of South Africa and of the South African Institute of Computer Scientists

Die amptelike vaktydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor

Professor D G Kourie
Department of Computer Science
University of the Pretoria
Hatfield
0083

Production
Mr Q H Gee
Department of Computer Science
University of the Witwatersrand
Johannesburg
Wits
2050

Subscriptions

The annual subscription is

	SA	US	UK
Individuals	R20	$ 7	£ 5
Institutions	R30	$14	£10

to be sent to:
Computer Society of South Africa
Box 1714 Halfway House 1685

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial

by

Derrick Kourie

It is my privilege to have been requested by the SAICS executive to take over the post of editor of QI from Professor Judy Bishop. I think it is in order to thank her on behalf of the readership for the fine job she has done in boosting the quality of the journal during her brief but effective term. It is also appropriate to thank the production editor, Quintin Gee, for his substantial role in producing the journal. I am grateful that he is still in the post, and for all the support and work that he continues to do.

My job as editor is directed towards the overall goal of serving the South African academic community in the various computer-related disciplines in particular, and the computer industry in general. A number of objectives which support this goal include

- ensuring that high quality papers are published, thereby providing a display window for computer-related research in South Africa
- boosting local and international circulation of the journal both within the academic community and in the computer industry at large, thereby promoting a fruitful interchange of ideas
- attempting to do this in a cost-effective fashion so that the limited financial resources of SAICS and the CSSA may be released (perhaps even modestly augmented) to promote their various other service-orientated activities.

A number of measures are planned which are intended to meet these objectives. I shall mention some of them below, while others will become manifest with the passage of time.

After much debate it has been decided to change the name of this journal from Quæstiones Informaticæ to The South African Computer Journal/Die Suid-Afrikaanse Rekenaartydskrif. It will be abbreviated to SACJ in English and SART in Afrikaans. Arguments against this name change include the conciseness and uniformity of reference in both official languages provided by QI, and a certain kind of catchiness to the name. Those in favour of the name change regard the new proposal as being more descriptive for ordinary mortals (i.e. non-Latin scholars), less pretentious, and therefore more inviting for a wider audience. The fact that the new title identifies the journal as South African is also regarded as important. Many readers would, I surmise, be fairly neutral about the name and adopt a philosophical "a rose by any name" position. Perhaps the divide is between those who opt for a high level of abstraction and information hiding, and those who feel that a measure of refinement is necessary.

Regarding the quality of papers, I shall continually strive to ensure that papers submitted are reviewed by at least two relevant and competent specialists. It is appropriate here to thank all those who have so enthusiastically reviewed papers to date. This is a time-consuming, altruistic, backroom task, with very little explicit reward. To ensure that the burden is spread more equitably, I would like to appeal to readers to suggest additional names of people who could be approached for reviewing. Names of overseas contacts would be particularly useful.

I should also like to invite as much reader-participation in the journal as possible. There are several levels at which this may be done. The most obvious is by way of letters to the editor. Many people out there have strong ideas about a variety of subjects. In the absence of a decent national network facility (perhaps someday!), please feel free to use SACJ as your soapbox.

However, it is also evident that many people read many books for a variety of purposes. Why not share these insights by submitting book reviews to the journal, particularly with respect to books which could be prescribed for courses? If there are any book publishers or distributors out there who perchance may read this editorial, perhaps you should make inspection copies to lecturers contingent on a review being provided to SACJ!

I would also encourage researchers to continue providing a steady stream of research papers to the journal. Clearly, SACJ is in competition with other international journals for your research results. However, this is not a head-on competition. While it would be sheer hubris to pretend that SACJ is precisely equivalent to one of the more prestigious overseas publications, there are considerations which argue in favour of submitting certain kinds of research to SACJ. First, SACJ will be dedicated to providing a quick turnaround in reviewing and publication. Hence, it is an ideal forum for presenting and testing interim research results, and even for quickly assuring your stamp on potentially important ideas which you hope to flesh out later. Secondly, SACJ is the obvious forum to use for locally relevant research. Finally, and quite candidly, the competition for publication in SACJ is obviously not as intense as in a more prestigious international journal. However, I need to be most

explicit on the implications of this latter point.

SACJ should not be seen as a soft option in the sense that quality will be sacrificed. By this I mean that on some arbitrary scale of quality measurement, if CACM contains papers above say the 95% percentile, then SACJ should fall into about a 60% percentile category. Put differently, there is clearly a gap to be filled that lies somewhere between poor, inferior drivel and outstanding research contributions – a gap which SACJ will seek to fill. Papers will therefore be rigorously reviewed, and every effort will be made to ensure that the journal is worthy of international recognition – even if such recognition does not come about immediately. This is not the impossible task that some might consider it to be. There are several South African scientific journals that already enjoy a measure of international recognition (the South African Statistical Journal – to name but one). Furthermore, it is my perception that many of our academics who travel overseas discover – perhaps slightly to their amazement – that they are well able to hold their own with academics at peer institutions. This suggests that there is probably sufficient brain power, research ability and research activity in the country to ensure that the goal of international recognition is attained.

As for the cost-effective functioning of SACJ, two points need to be made. First, SACJ will be available for a limited amount of advertising at R1000 per page and R500 per half-page. The computer industry and book publishers might wish to avail themselves of this offer, as might universities and employment agencies. Enquiries in this regard should be directed to Quintin Gee. Secondly, a modest charge per page (indicated elsewhere in this edition) will be levied on accepted research papers. This has become standard practice for most journals, the rationale being that the SACJ is one of the journals which counts for state subsidy purposes. However, the editor will have the right to waive such charges in deserving cases, as for example in the case of an author from industry whose company is unwilling to provide the financial support.

Ultimately then, SACJ will critically depend on your support. It will become what you, the reader, researcher and reviewer, make it. In a sense the South African Computer Journal will expose you, the South African Computer Academic, to the outside world without a single Latin phrase to hide behind.

Reflections on the Nature and Future of Computer Science in Southern Africa

B H Venter
Department of Applied Computer Science, University of Fort Hare, Private Bag X1314, Alice, Republic of Ciskei

Received September 1988, Accepted October 1988

1. Introduction

Computer Science is a relatively recent scientific discipline. It has seen rapid growth and even more rapid change. Not surprisingly there is constant controversy among computer scientists about the nature of the discipline, and how best to pursue it.

This article aims to stir up the controversy. The views expressed in it are personal, and the article is an adaptation of the author's inaugural address at Fort Hare University.

2. What's in a Name?

Before going on to more serious matters, I wish to reflect on the name "Computer Science". The name has been the subject of one of the most heated and least productive controversies, and Fort Hare has not escaped.

To get some idea of how low the level of debating can become, consider the following statement which one hears far too often

> "If a discipline has to attach the word science to its name, it's probably not a science. For example:
>
> Chemistry, Physics, Geology, etc.
>
> as against
>
> Social Science, Political Science, Computer Science."

When confronted with such a statement many Computer Scientists react by coming up with names like Computerology, Informatics, and so on. I believe this is misguided because:

- Changing the name of the discipline is not going to change its nature.
- It is terribly narrow-minded to contend or accept that the classic disciplines of the natural sciences are the only truly "scientific" disciplines.

Of course, for the most part, people wish to change the name of the discipline because they believe that it does not accurately reflect the nature of the discipline. Hence one often hears names like Computing Science, Programming Science, Information Science, Information Processing, Software Engineering, and so on.

It also happens that a university department changes or extends the term "Computer Science" in order to indicate that it is somehow different or superior to the "plain vanilla" computer science departments. For example, the Fort Hare department is currently known as the department of "Applied Computer Science." I suspect that it acquired this name in order to place some distance between the curriculum of the department and the very Numerical Mathematics oriented curricula of departments at other universities in the early seventies. The distinction has since faded considerably.

I do not believe that playing around with the name of the discipline or an individual department serves any useful purpose. The term "Computer Science" has become entrenched and is not likely to be changed. Virtually every computer science department in Southern African is called "Department of Computer Science" and I believe Fort Hare should follow suit.

3. Is Computer Science a Natural Science?

A less heated but far more serious controversy concerns the fundamental nature of Computer Science. Is it a natural science, or an engineering discipline, or the underlying discipline of a profession distinct from Natural Scientist or Professional Engineer? Put in another way: does a computer science department belong in a natural science faculty, an engineering faculty, a commerce faculty, or a faculty of its own?

Put in yet another way, should a South African professional computer scientist be registered in terms of the act covering natural scientists or the act covering professional engineers, or should there be a special act, or should there not be any act at all?

In the United States computer science departments have ended up in natural science faculties as well as in engineering faculties, depending on whether the "parent" department had been Mathematics or Electrical Engineering. Apparently, these departments flourish equally well in either faculty.

In South Africa, most departments find themselves in natural science faculties. Correspondingly, the

South African Council for Natural Scientists (SACNAS) attempted to list "Computer Scientist" as one of the professions covered by its act.

On the other hand, SACNAS did not, at that stage, recognize the South African Institute for Computer Scientists (SAICS), and hardly consulted SAICS about the listing. Officially this happened because SAICS did not have enough qualified members when it sought recognition. However, one cannot help but suspect that SACNAS did not really wish to find a way around this difficulty. Perhaps, some of the senior natural scientists involved in the issue were not quite convinced that Computer Science is a natural science.

The intended listing of "Computer Scientist" as a profession reserved for registered natural scientists was met by an outcry from the myriad of non-graduates who make a living from programming computers. Few of these people would qualify for registration as natural scientists and they quite rightly fear that listing "Computer Scientist" may eventually lead to their exclusion. Of course, the employers of these people are even more horrified at the prospect of having to compete for the services of a very small pool of academically qualified people.

Needless to say, the masses prevailed and "Computer Scientist" was removed from the list of professions reserved for registered natural scientists.

My own opinions about these issues are:

- Computer Science may well not be a true "natural science" because it is not chiefly concerned with natural phenomena, However, if that is the case, then one should exclude the Mathematical sciences as well.
- Computer Scientists ought not to register with SACNAS, and SACNAS probably ought not to exist in the first place. The computer software industry, for one, is not yet ready for a limited entry profession, and public safety will be better served by making software producers more accountable for the quality and consequences of their products.
- I personally feel most comfortable in a natural science faculty, and students from such faculties tend to fare better with Computer Science than students from other faculties. Moreover, in Southern Africa, the structured curricula of engineering and commerce faculties are simply too crowded to adequately accommodate the topics a professional computer scientist should know.

  I feel further more that the difference between Computer Science and Physics is no greater than the difference between Physics and Zoology or between Zoology and Chemistry. And, on the whole, I feel that Computer Science is more similar to the natural sciences than to any other grouping of sciences.

4. Do we need Computer Scientists in Southern Africa?

Computer science departments largely base their curricula on a model drawn up in the United States, and succeed in producing graduates that are internationally acceptable as computer scientists. This is no empty claim, for quite a few of our graduates proceed overseas to further their studies, and they generally do quite well there. Unfortunately, some never return.

One of the reasons for this mini brain drain may well be a lack of jobs for computer scientists (A "Computer Scientist", of course, is someone with at least an honours degree). Certainly, many of the industries that employ computer scientists in developed countries have no local equivalents. Instead, the bulk of computer related employment in Southern Africa is offered by the data processing departments of companies that have little interest in computers other than using them to implement business information systems.

In the past, the management of some of these data processing departments have been sharply critical of the curricula of computer science departments. In effect, their criticisms amounted to the following : local industry has very little use for computer scientists, and instead needs commerce graduates with a sound computer background, augmented by non-graduate programmers trained with specific programming skills.

Such complaints and representations have led to the creation of departments of "Business Data Processing" in the commerce faculties of universities. Furthermore, Technikons and private training companies have rushed in to provide training to the legions of non-graduate programmers demanded by data processing departments.

Such developments appear to eliminate data processing departments as a reason for producing computer scientists. What's left of local industry is mainly the growing number of companies that develop complex systems incorporating computers. These companies certainly have more use for computer scientists. However, they are typically dominated by engineers and their managers claim to need "software engineers" rather than computer scientists.

Not surprisingly, engineering faculties have rushed to include programming courses in their curricula, and they are starting to claim that software engineers should be produced by engineering faculties rather than by science faculties.

One may thus reasonably ask whether it makes sense to go on producing computer scientists when those that do not emigrate end up with employers that supposedly have little use for computer scientists. Would it not make more sense to distribute the handful of computer science lecturers

among engineering and commerce faculties, and forget about having computer science departments?

My view is that the following reasons are more than adequate to justify the existence of computer science departments in their current form:

- A BSc with Computer Science as a major subject is a useful education even for students that do not go on to become active computer scientists.
- Computer Science, in its present "scientific" form, is a sensible minor subject for any bachelor's degree, since it thoroughly demystifies the computer. It is also an effective way of developing problem-solving skills, methodical thinking, and the ability to abstract.
- Constructing computer software is an extremely demanding task. Much of what is now done by non-graduates should rather be done by computer scientists. Employers are starting to realize this.
- The curricula of engineering and commerce faculties are too crowded to allow their students to acquire all the skills and knowledge needed to become effective software developers.
- Eventually, the availability of professional computer scientists will lead to industry undertaking more projects that cannot possibly be carried out by non-computer scientists.

5. The State of Computer Science Research in Southern Africa

Coming back to the computer scientists that further their studies overseas and then stay there. I guess that at least a few of them become computer science researchers, and stay overseas because they despair of their chances to do research in Southern Africa. Certainly, it is true that university lecturers are expected to do research, and many *do* research, but it is also true that lecturers get very little time for research and very little recognition for what they manage to do.

And while it is true that most computer science departments are understaffed, this does not mean that a prospective researcher can easily find a suitable job upon graduating. Right now, Fort Hare would probably be unable to hire such a graduate.

To the best of my knowledge, no computer science research is currently being conducted outside of universities. It may well be that I am simply not informed about it, but it is also certain that such research is neither published in the research journals nor publicized elsewhere.

Until a year ago, the CSIR maintained a computer science department as part of the, now defunct, National Research Institute for Mathematical Sciences. This department, while under-equipped and understaffed by international standards, had a staff including more than 10 professional computer scientists (including myself), most of whom were pursuing higher degrees.

This made the department larger than any university department apart form UNISA. Correspondingly, almost a third of all the papers presented at the most recent South African Computer Symposium originated from this department.

Unfortunately, the publishable research conducted by this department was severely impeded by management attitudes and priorities, as well as by the lack of senior staff of international stature. The department was thus unable to establish computer science research on a firm and respectable footing in Southern Africa.

Nevertheless, while it existed, the department had the potential of doing this. Unfortunately, the demise of the CSIR as a body carrying out research with its own staff has squandered this potential. The newly appointed chairman of the governing council of the CSIR recently motivated this as follows (my translation):

> "Initially this (research carried out by the CSIR) was the best approach, because South Africa needed a strong scientific infrastructure.
>
> Now, however, we have reached the stage where the universities can truly stand on their own, and the basic sciences are firmly rooted.
>
> The CSIR thus need not duplicate what the universities are doing, or are able to do, and can rather use its enormous pool of talent in the service of industry."

Following this change of strategy by the CSIR, many of its computer scientists (including myself) left its service. Those that did not, or could not, were transferred to the CSIR computer centre (data processing department) and told to earn money somehow. I doubt whether much publishable computer science research will ever again originate from the CSIR.

Presumably, the current state of computer science research at the universities, had little influence on the making of such momentous decisions, In fact, computer science issues probably do not figure in many decisions at all. The low status of computer scientists is perhaps best summarized in the introduction to the proceedings of the last South African Computer Symposium, written by Prof. Pieter Kritzinger of UCT:

> "I know of no computer scientists in South Africa who is in a position where (s)he can affect funding priorities. As far as I know we have no representation on any of the committees of the Foundation for Research and Development, and I know of no-one in our Afrikaans speaking fraternity who is a member of the Akademie vir Wetenskap en Kuns. It will take time and conscious effort to establish our presence. The same is true of course for our universities. Again, with one exception, I know

of no dean of a science faculty, vice-principal or principal who is a computer scientist."

As I have already mentioned, computer science lecturers have little time available for research. This problem is greatly compounded by the very nature of computer science research: all computer science research is ultimately concerned with the construction of software. More often than not, a research project requires a great deal of software to be specially constructed, and constructing software is extremely time consuming. The result is that years of work may culminate in a single research paper. In fact, in reading research in my fields of interest, I seldom encounter the same author twice, except when the same paper is published over and over again.

Coupled to the low stature of computer scientists, this trickle of papers per individual researcher makes it almost impossible, if not completely impossible, for a computer scientist to obtain significant funding from the Foundation for Research and Development, in competition with established researchers in other fields, who in some cases manage to produce as many as ten papers per year.

The result is insufficient equipment, insufficient library allocations, insufficient opportunities to attend research conferences, insufficient post-graduate students, insufficient research-oriented computer science lecturers, and so on. Which in turn perpetuates the insufficient research. I regret to conclude that computer science research is in a sorry state in Southern Africa. Moreover, the signs are that things are getting worse rather than better.

6. Do we Actually need to do Computer Science Research in Southern Africa?

One may justifiably ask if it really matters whether or not computer science research is carried out in Southern Africa. Lecturers need to do research in order to be able to train research students. But if these students cannot find jobs as researchers, apart from overseas or as lecturers, why go to all this trouble? Why not simply educate students up to the honours level and be done with it?

Well, one obvious answer is that one is going to have trouble finding and keeping university lecturers if this is the attitude taken, and this may already be happening. But in addition to that, I feel that a thorough training in basic research is an excellent background for doing development and implementation.

That is, we may not need computer science researchers outside of the universities, but we do need people that were trained as computer science researchers. Besides, the availability of enough trained researchers may in itself lead to the creation of a need for researchers.

7. So what do we do about it?

One is always tempted to pronounce "the central government must do something about this, or there will be dire consequences to the national well-being." And certainly, in the case of South Africa, one may reasonably ask whether a country that can support an Antarctic research team, and that is contemplating a space program, will not be better off sacrificing one of these in favour of comprehensively supporting computer science research.

However, from the bleak picture I have painted thus far, I think it is clear that public sector and private sector policy makers are not likely to come to the aid of Computer Science in Southern Africa.

It is thus up to policy makers in the universities to provide the necessary support for Computer Science. It is largely thanks to the efforts of these people that Computer Science was established in Southern Africa in the first place, and it will largely be up to them to keep it viable in the future.

Of course, it is ultimately up to the computer scientists themselves to do so well with *is* available, that the discipline will ultimately generate its own growth, funding and respectability.

Abstracts: MSc/PhD Conference held at Dikhololo in 1988

The 3rd Conference for MSc/PhD computer science students, held at Dikhololo in 1988, was attended by representatives of most South African Computer Science departments. The conference, which was sponsored by the RCP Group and organised by the Computer Science Department of the University of South Africa, was worth attending, and included some very interesting research activities. Abstracts of the papers presented at the conference are produced below.

The Specification of an Operating System

H C Ackerman
University of Stellenbosch

Abstract

Computer hardware has improved dramatically over the past decade. Current technology offers processing potential on a chip which was traditionally associated with mainframe computers. Software science development has unfortunately not kept up with engineering progress, and the development of systems today is handicapped by software reliability rather than hardware.

The correctness of large software systems depends on three main factors: correct decomposition of the system into modules; individual correctness of the modules; and correctness of the interfaces between component modules. This paper is concerned with the latter: formal specification of module interfaces.

Ideally a system should be decomposed into component modules which are implemented in parallel by several programmers. Experience shows that errors are caused by the integration of modules, indicating that the formalism used for the specification of decomposed models is inadequate. It should be possible to check the correctness of a specification mechanically in order to rule out human error in the proof of specification correctness.

Algebraic specification offers a formal method of specifying systems top-down, and several specification languages for algebraic specification are in use today. We examine the feasibility of a comprehensive algebraic specification of a non-trivial operating system.

A Semantic Interpreter for LOTOS

D Behr
University of Pretoria

Abstract

System specifications for communication and distributed systems using FDTs like LOTOS have been receiving a lot of international attention. These systems tend to be concurrent, asynchronous and non-deterministic. Specifying and verifying such systems using traditional techniques is difficult. As part of a research project at UP, an environment for the specification of communication and distributed systems is being developed. The part of the project discussed here deals with the development of a semantic interpreter for LOTOS specifications. The semantic interpreter generates traces for a specification. These traces provide a means of verifying the LOTOS specification. International attention tends to prove a trace as either acceptable, or not acceptable to a specification. This approach however tends to provide another means of characterizing a system, by generating a sufficiently big subset of traces and/or subtraces. These provide an alternate means of verifying aspects of a specification.

The Feasibility of Applying Expert System Technology to the System Performance Analysis and Tuning of Operating Systems

D A Bryant
University of Stellenbosch

Abstract

It is proposed that an expert system that can be applied to analyse the performance of IBM mainframe computer systems running under the OS/VS2 MVS operating system should be developed. The expert system will utilize performance data logged by the standard MVS software monitors such as SMF, RMF, GTF and the CICS monitoring facility. The expert system will analyze this data and report on the inefficient usage and performance degradation of the system. The expert system will identify exception events and provide the most likely cause of the event, together with the actions recommended to remedy the situation. The knowledge embedded in such a system will represent the skills of the most expensive and scarce programmers in the computer industry.

In order to evaluate the feasibility of this approach, three prototypical expert systems have been developed. Each assesses the ASM or Auxiliary Storage Management component of the OS/VS2 MVS operating system. The placement and size allocation of the data sets on the DASD devices within the system configuration are evaluated, possible system bottlenecks are highlighted, and recommendation to improve system performance are made.

The first expert system developed makes use of Goldworks, an expert system development tool, or expert system shell based on Golden Common Lisp software. The second uses the environment provided by Arity Prolog, an extended version of the Prolog programming language. The final expert system has been developed using a procedural language, Modula-2, to mimic the actions of the AI languages.

This paper evaluates the relative applicability and suitability of each of the three approaches to the particular problem area under consideration.

Speeding Up Ray-Tracing

R F Breedt
CSIR

Abstract

Ray-tracing is an elegant approach to realistic image generation. Shadows, reflections, transparency and other visual effects can all be handled in an integrated way. However, being a brute force method, ray-tracing is a very slow process. Speed-ups can be achieved by exploiting the inherent parallelism of the process and by using approximate data structures.

This paper introduces the concept of ray-tracing and then discusses various software and hardware solutions for reducing the time complexity of ray-tracing.

Image Processing Applications on the Transputer

N Cooke
Rhodes University

Abstract

This paper describes an investigation into the application of Image Processing techniques on a transputer network hosted by a PC-AT. The research concentrates on identifying an appropriate distribution of tasks between the network and the 80286 processor for Image Processing applications. The major considerations in the investigation are benchwork of standard Image Processing techniques

on the two processing systems and the comparison of their architectural features which support Image Processing. To complement the investigation, a system has been developed for displaying images from either of the processors with the aid of an Enhanced Graphics Adapter. The results of the research are used to draw conclusions about the viability of this hardware configuration as an Image Processing workstation.

EXPROG: 'n Outomatiese Programmeringstelsel

J P du Plessis
Universiteit Oranje Vrystaat

Abstract

EXPROG is 'n outomatiese programmeringstelsel wat Pascal programme genereer. EXPROG maak gebruik van die kennis-gebaseerde benadering tot probleemoplossing. Gevolglik besit EXPROG 'n verskeidenheid soorte (tipes) kennis wat deur middel van verskeie voorstellingsmetodes, bv. reëls, rame en EXPROG definisies, voorgestel word. Die gebruiker beskryf, in Engels, die spesifikasie van die verwagte program aan EXPROG. EXPROG redeneer dan met behulp van sy bestaande kennis totdat 'n algoritme gekonstrueer is wat voldoen aan die spesifikasie soos verskaf. Daarna word die algoritme gekodeer in Pascal om sodoende die verwagte program daar te stel.

A Modula-2 Implementation for the Transputer

W J Hayes
CSIR

Abstract

This paper reviews message passing techniques for communication between processes. An implementation of Modula-2 for the Transputer is then given. Finally, added features to Modula-2, providing concurrent programming and message passing facilities, are discussed.

Analytic Modelling Approaches for DASD Subsystems

H Joubert

Modelling of Shared Resources in Computer Systems: The Multiserver Station with Concurrent Customer Classes

S Crosby
University of Stellenbosch

Abstract

With the sponsorship of the SA Post Office and Semantyx (Pty Ltd.) a group of students at the University of Stellenbosch is undertaking research into the modelling of DASD subsystems of large computer systems. Research involves the investigation, application and development of appropriate mathematical techniques to model DASD subsystems to a high degree of accuracy. The group will produce a modelling package with a graphics interface based on existing windowing software, which will allow easy specification and modification of models.

Performance monitoring data from the DASD subsystem to be modelled will be used for automatic

construction of the model, and results of the model will be displayed graphically.

Two papers will be presented, which will give an overview of the project and cover in some detail the existing techniques for, and proposed research into, mathematical modelling of DASD subsystems.

A Survey of Standards for the Exchange of Digital Geographical Information

A Cooper
CSIR

Abstract

There are a number of countries and international organizations who are developing standards for the exchange of digital geographical information. Some of these standards attempt to cater for all types of geographical information while others are aimed at specialized subsets of geographical information, such as cadastral or topographic information. While some standards have been implemented, none of them have been tested completely.

The author has been involved in drafting the South African exchange standard and has held discussions with people involved in the drafting of the standards of the United States, the United Kingdom and the International Hydrographic Organization. This paper will discuss these and other exchange standards.

Parallel Process Placement

C Handler
Rhodes University

Abstract

This paper investigates methods of automatic allocation of processes to available processors in a given network configuration.

The major considerations which have been addressed in deciding upon a particular allocation scheme are:

- the number of processes to be allocated versus the number of processors available in the network;
- the computing power of each processor;
- the work load of each process; and
- the interdependency between processes, e.g. the use of shared variables and the degree of inter-process communication.

The research work involves the implementation and testing of various algorithms for optimal process allocation, as well as the gathering of performance statistics during program execution for use in improving subsequent allocations.

The system has been implemented on a network of loosely-coupled microcomputers using multi-port serial communication links to simulate a transputer network. The concurrent programming language Occam has been implemented replacing the explicit process allocation constructs with an automatic placement algorithm, enabling the source code to be completely separated from the hardware considerations.

A Knowledge-based Design Environment for Information Systems

J Kambanis
University of South Africa

Abstract

This paper describes the functionality and architecture of an Information Systems design environment which is based on a Semantic Data Model. Recent research in Software Engineering has recognised the promising possibilities of using Knowledge Representation techniques from Artificial Intelligence, in

modelling Information Systems. The environment described is based on this approach. It integrates User Interface techniques used in Computer Aided Software Engineering and Knowledge Acquisition Environments to support modelling of Information Systems. Knowledge representation is based on the Taxis language development at the University of Toronto. The architecture of the environment is aimed particularly at maximising its generality and extensibility.

The Synthesis and Processing of Waveforms to Generate Musically Interesting Sounds

A J Kesterton
Rhodes University

Abstract

Audio frequency sound of musical interest can be generated by converting a set of data from the digital to the analog domain. Various standard methods exist to generate the data and alter it to create more musically interesting sounds. These methods can be implemented in hardware or software or both.

This paper gives details of Frequency Modulation and Additive Synthesis algorithms being implemented entirely in software to generate sound data. The implementation of various filters and the direct manipulation of the digital data are also discussed. The downloading of the resultant data, using the MIDI protocol, to a sampler (a type of digital music synthesizer) to perform the conversion to the analogue domain is described.

The software is designed to run both as a stand-alone system and as part of a Music Network system being implemented at Rhodes. The Network uses the XINU real-time operating system and Ethernet hardware on IBM PC look-alikes.

Sellulêre Outomate

L Kotzé
Randse Afrikaanse Universiteit

Abstract

Een- en twee-dimensionele sellulêre outomate word gedefinieer. Verskillende eienskappe van sellulêre outomate word ondersoek; onder andere die volgende:

- Die patrone wat voortgebring word deur die outomate onder verskillende begintoestande.
- Die ooreenkomste tussen twee- en meer-toestand outomate. Die ooreenkomste tussen sellulêre outomate en PASCAL se driehoek.
- Die afledingsvermoë van sellulêre outomate.

Toepassings vir sellulêre outomate in die fisika, chemie en ander velde word bespreek.

An Expert System for Diagnosing, Tuning and Optimising Computer Systems Performance

W H Kotze
University of Stellenbosch

Abstract

The expert system VAXEXP evolved from the need to support the systems and performance management of large and complex VAX systems. There was also a need to provide some training support for junior systems staff.

The initial starting point is the VAX system(s) where a combination of Fortran, VAX DCL and System Services collects the required data and concentrates it into one large ASCII file. This is ported across to a microcomputer supporting Arity

Prolog under MS/DOS 3.1. VAXEXP is planned to consist of three main modules, the first of which, called SYSGEN, will provide information on the usage of the systems parameters, indicate the implications of the value of a particular parameter, and lastly uses the data from the VAX to suggest a better suited value for a parameter. This module is nearly finished and contains some 350 rules.

The second main module, called PERFLEARN, is designed to be an aid in building up expertise that will enable VAXEXP (and the system manager) to turn a qualified statement into a quantified rule, e.g. "the direct I/O is EXCESSIVE" → "If the direct I/O is xxxx THEN it is EXCESSIVE".

The final module, called PERFEVAL, will actually make extensive use of the data from the VAX to identify and diagnose existing and potential problems and where possible will also suggest a remedy. There are some 500 potential rules for this module.

Query Processing in Distributed Database Systems

S M Lamprecht
University of Stellenbosch

Abstract

Query processing in distributed database systems involves the determination of an execution strategy. An execution strategy is a sequence of operations that must be performed on the database in order to obtain the required result to solve the query. For a given query several execution strategies can be ascertained. It is therefore a very important problem in query processing to acquire the optimal execution strategy. An execution strategy is optimal with respect to a selected cost function. This cost function is determined by several different factors. One of the most important factors in the distribution environment is the cost of transmitting data between different computers. Several optimisation techniques exist which optimise data transmission cost. Some of these optimisation techniques will be presented.

Exploiting Redundancy in Knowledge Representations

R Layton
University of the Witwatersrand

Abstract

Most complex representational formations embody both explicit and implicit redundancy. Much effort has been extended at both identifying this redundancy and reducing it to some minimal form.

Isolated cases have proposed that redundance be exploited within specific areas of knowledge such as machine vision and sensor-based diagnostics but no overall model has yet been proposed.

A model is proposed that exploits both implicit and explicit redundancy in order to maximise consistency-checking and to facilitate multiple viewpoints in a single representation. This leads to a conclusion that, whereas a universally accepted representation cannot exist, a hybrid model with controlled redundancy exhibits properties required of such a universal representation.

A Communication Kernel for a Distributed Database System

M D Meumann
University of Stellenbosch

Abstract

The major components of a Distributed Database Management System comprise, amongst others, a network facility enabling access to remote sites and a concurrency control mechanism to handle and

synchronize distributed transactions. These two features are the major components of the communication kernel of a DDBMS currently under development at the University of Stellenbosch using the XENIX operating system. This talk will describe the networking features being included in the XENIX operating system using ArcNet communication cards and the concurrency control being implemented based on a conservative timestamp algorithm.

An Algorithm Development System

C C Pienaar
University of Stellenbosch

Abstract

A short informal introduction is given to the weakest precondition methodology for developing an algorithm and a proof of its correctness, with reference to other popular programme verification methods. An example of algorithm development using weakest preconditions is presented, with comments on the role an interactive, computerized development system could play in this process. Finally, an overview is given of some of the problems encountered in automating such a methodology.

Parallelle Logika Programmering

A E G Potgieter
University van Pretoria

Abstract

Daar is verskeie benaderings tot die parallelle verwerking van logikaprogramme waarvan EN-parallelisme, STROOM-parallelisme en OF-parallelisme die belangrikste is. Tydens EN-parallelisme kan datakonflikte ontstaan, en gesofistikeerde kontrole word benodig om hierdie konflikte te verhoed.

Huidige navorsing in die gebruik en implementasie van parallelle logika tale sal bespreek word met spesiale verwysing na die implementasie van EN-parallelisme in J S Conery se EN/OF prosesmodel, en die implementasie van STROOM-parallelisme in PARLOG, Concurrent Prolog en GHC.

'n Stelsel wat ontwikkel is op 'n netwerk van transputers, waar STROOM-parallelisme verkry word deur pyplyning, sal verder bespreek word.

Formalisation of the Entity-Relationship Model

H H Rennhackkamp
University of Stellenbosch

Abstract

The entity-relationship data model was originally presented by Chen; and many extensions and improvements have since been proposed. It is an improvement on the relational data model in that data is viewed in the more natural manner as consisting of entities and the relationships between them, each with their descriptive attributes.

The relational data model has been formalised in many ways; from the original formalisation by its presenter Codd, in terms of data objects, operations and integrity rules. This formalisation has since been improved a number of times; also by Jacobs in terms of an extension of first order logic.

A similar extension of first order logic is presented as a formalisation of the entity-relationship data model.

Computer Animation and Applications

C F Scheepers
CSIR

Abstract

Developments in low-cost high-end graphics workstations have placed the many applications of computer animation within our reach. A combined use of these workstations with current video technology provides enough "hardware" capabilities to do animations. Animation software systems developed overseas are also available, thus only a lack of funds and possibly of imagination now prevent those interested to create their own computer generated animation sequences.

This paper will discuss computer animation and associated problems. Various applications of computer animation are also mentioned, specifically in education, entertainment, advertising and visualization in science. The discussion will be highlighted with a computer animation presentation.

Intelligent Computer Aided Instruction

T Thomas
University of South Africa

Abstract

The aim of intelligent computer aided instruction is to create systems that can teach and adapt to students on a one to one basis as a human teacher can in the same situation. Each teaching step is not hardcoded into the system but is decided upon based on the student's model at a particular time and the work that has been done previously. This paper discusses intelligent computer aided instruction in general and refers to a proposed system the author hopes to build to implement some of the ideas of intelligent tutoring. The first part of the paper discusses what intelligent CAI is and how it differs from conventional CAI. A discussion of the use of the overlay method for student modelling follows with reference to the work that has been done and the work proposed. The use of artificial intelligence techniques in the creation and use of teaching strategies based on the overlay model is then discussed with a description of some of the work planned in this area. The necessity of an authoring system that can create intelligent systems is then introduced.

Distributed Database Recovery

H Viktor
University of Stellenbosch

Abstract

The NRDNIX-system is a distributed database management system prototype currently being developed at the University of Stellenbosch. One of the main advantages of a distributed database over centralized counterpart is that nonfailing sites continue operating in the event of other site failure(s). This complicates the recovery process and it is therefore essential that robust recovery techniques must be implemented that ensure the global consistency of the database at all times. This is achieved by combining local recovery techniques and additional methods developed to facilitate distributed recovery.

In NRDNIX data is dynamically duplicated as needed. It is essential that these multiple copies have to be kept consistent even in the event of failure of one or more sites. This further complicates the recovery process and emphasises the need for the provision of a globally consistent view of the database at all times.

An overview of the factors which have to be brought into consideration when developing a

globally consistent system is given, as well as the recovery techniques implemented to provide a globally consistent distributed database.

Research on Intelligent Computer-Assisted Instruction (ICAI)

J van den Berg
University of Pretoria

Abstract

ICAI research lies at the intersection of the domains of artificial intelligence, cognitive psychology and computer aided instruction. The aim of ICAI is to develop adaptive instructional systems based on a diagnosis of the student's learning needs. Current ICAI systems use AI programming techniques and are usually implemented in languages such as LISP and Prolog. These systems usually consist of

- a problem-solving or expertise module
- a student model
- a tutorial module
- a communication module

Due to the size and complexity of ICAI systems, most current systems only focus on one or two of the above mentioned components.

The aims of this research are

- to critically evaluate the theoretical base, instructional principles and development process used in current ICAI systems
- to identify related areas of study from which known principles can be used in the development of ICAI systems
- to propose a methodology for designing and implementing ICAI systems as complete instructional systems
- to implement aspects of the research in a practical ICAI system
- to identify an area of ICAI (or a related topic) for PhD studies.

An Overview of Image Processing Environments

J Smit
CSIR

Abstract

The last two decades have seen image processing systems migrate from mainframe to workstation and micro to be used for an ever-increasing range of applications. This paper gives a brief overview of image processing applications and techniques. It traces the development of image processing environments, discusses some of their design issues and finally describes the functional components of an image processing environment.

A Frame- and Rule-Based Approach for Developing Medical Expert Systems

J van Jaarsveld
CSIR

Abstract

To date most local expert systems use the rule-based approach and few incorporate the frame approach. This paper describes the development of a medical expert system for interpreting the effects of hypertension using a combination of frames and rules. The structure and properties of the system are described by using the Arity Prolog/Expert Development System. In addition, the inference process linking the frames and the rules is discussed. Finally, some aspects of the knowledge engineering experience are detailed.

Determining Truth Values of Quantified Statements

A P Viljoen
University of South Africa

Abstract

In classical logic there exists a standard procedure to follow when one wants to determine the truth value of a given formula in a specific interpretation.

A new procedure, equivalent to the standard one, is proposed. This procedure also allows the calculation of truth values for formulas containing fuzzy predicates. A fuzzy predicate is interpreted as a fuzzy subset of a universe of discourse. Limiting the interpretation to a finite set, the quantifiers "for all" and "there exists" as well as nonstandard quantifiers like "few", "many" and "most" can be handled in a uniform manner.

The Application of Real-time Design Techniques to Simulation

G C Wells
Rhodes University

Abstract

This paper discusses the application of the Ward and Mellor design methodology for real-time systems to the field of simulation. The tools and heuristics used by Ward and Mellor are extended to provide a useful methodology for the design of real-time simulations. This is illustrated by the example of a real-time simulation of a manufacturing plant and process control system.

Software Design to Meet Third World Requirements: An Experimental Software Engineering Approach

Philip Machanick
Department of Computer Science, University of the Witwatersrand, Johannesburg, 2050 Wits

Abstract

Appropriate technology refers to technology appropriate for use in less developed parts of the world, especially the Third World; this paper raises some problems in adapting a definition of appropriate technology to computer software. A partial solution, a strategy called experimental software engineering, is introduced. The potential of this solution is demonstrated by a case study, in which software for medical education is developed. The result is a clearer understanding of both appropriate technology and design of software for usability.
Keywords: appropriate technology, software engineering, human-computer interaction, medical education

Received September 1988, Accepted January 1989

1. Introduction

Computers are becoming increasingly widely used in technologically developed countries, such as the USA. An important part of this trend is the migration of computers to contexts where users are technically naïve. This move of computers to the mass market has yet to impact less developed parts of the world. For example, Gambia has been reported as having no computer dealers at all in 1987 [5].

This paper examines some problems in fitting computers to needs of third world countries, from the starting point that dropping prices must eventually overcome the problems such as lack of foreign capital which inhibit the distribution of computers to poorer parts of the world. The fact that computer hardware is becoming more affordable does not mean computers will meet the needs of a third world country: the needs of the new society should be taken into account.

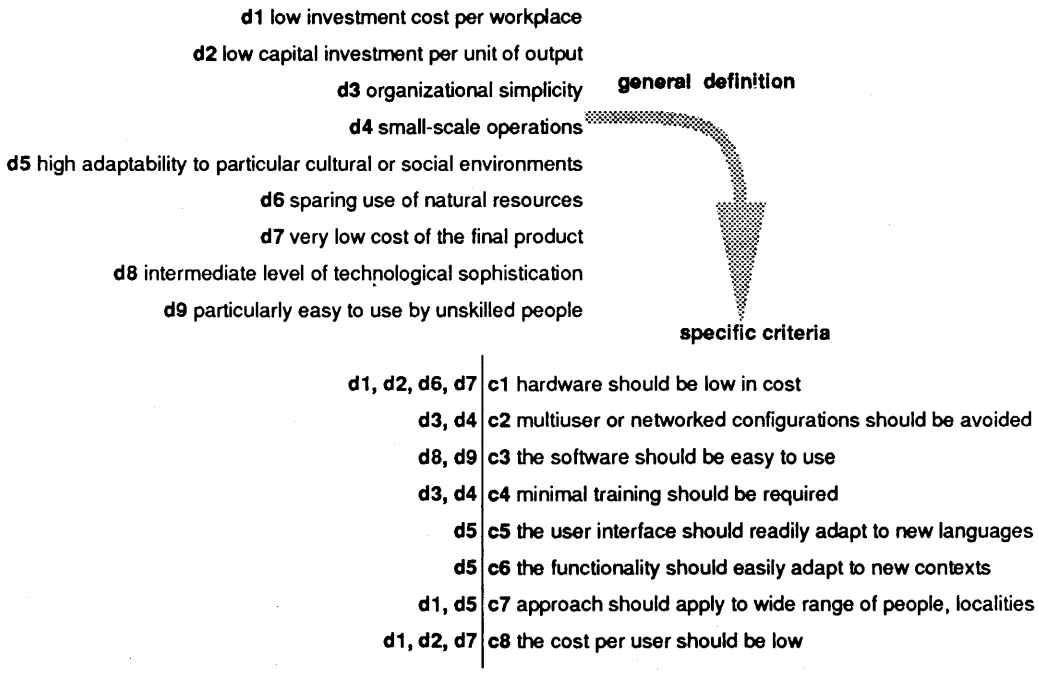
Much research into adapting other technologies to the third world has already been undertaken [9, 16]. Appropriate technology is the name broadly given to technologies considered suited to less developed societies. Mostly, "appropriateness" is measured by simplicity, and avoidance of high capital costs. Where computers are concerned, some thought needs to be given to the definition of appropriateness, since computers are a relatively advanced technology yet they are increasingly becoming affordable. In order to simplify the problem, the issue addressed here is how computer technology may be made to fit a specific, accepted definition of appropriate technology. This paper is further restricted in its scope in looking at a case study of a single application. In addition, the reasons for choosing the application domain are not presented.

The major focus of the paper is introducing and evaluating a strategy called experimental software engineering (ESE). ESE is a strategy for deriving the requirements when they are not clear, especially when the users have had little exposure to computers. ESE emphasizes usability, and placing the user in control of both the requirements and the finished product.

The next section looks in more detail at how a definition of appropriate technology may be applied to computers. In particular, criteria for deciding whether a computer approach is appropriate technology are presented. The following section introduces ESE, and explains how it relates to the problem of meeting the criteria for appropriate technology. From this start, a case study, in which the requirements for a tool for medical education are developed, is presented to illustrate how ESE may be applied. This case study is used to demonstrate how ESE brings out issues of usability, and gives the user control of the development process. In conclusion, ESE is evaluated in terms of appropriate technology concerns, and lessons which apply more generally to software engineering are considered.

2. A Definition of Appropriate Technology

Appropriate technology is technology specially adapted or designed for less-developed (especially third world) countries. Mostly, research into appropriate technology has revolved around low technology industries and agriculture. It is not obvious how experience of this kind may adapt to a relatively sophisticated technology such as the computer. For this reason, some attention is given to adapting an existing definition of appropriate technology to this new area.



A definition of appropriate technology [9] is used to derive criteria directly related to computer applications.

Figure 1: Criteria for computers as appropriate technology

A specific definition, presented in figure 1, is used as a starting point. This section derives criteria for evaluating computers as appropriate technology from this definition. Derivation of these criteria as illustrated in figure 1 is not presented in detail here (see [13]), since the major thrust of this paper is a description of the software development strategy employed. Instead, the implications of the criteria are considered here.

The first criterion (c1) – that the hardware should be low in cost – is derived from no fewer than four points of the definition, yet it is considered the least restrictive. Computer hardware is continually dropping in price; today's expensive workstation is tomorrow's personal computer. Even in an area as sophisticated as artificial intelligence (AI), tools for low-cost equipment such as IBM PCs and Apple Macintoshes are beginning to rival those on research machines of the last decade [12]. Low cost, then, should be seen as relating to a specific project, rather than as an inhibiting factor on research.

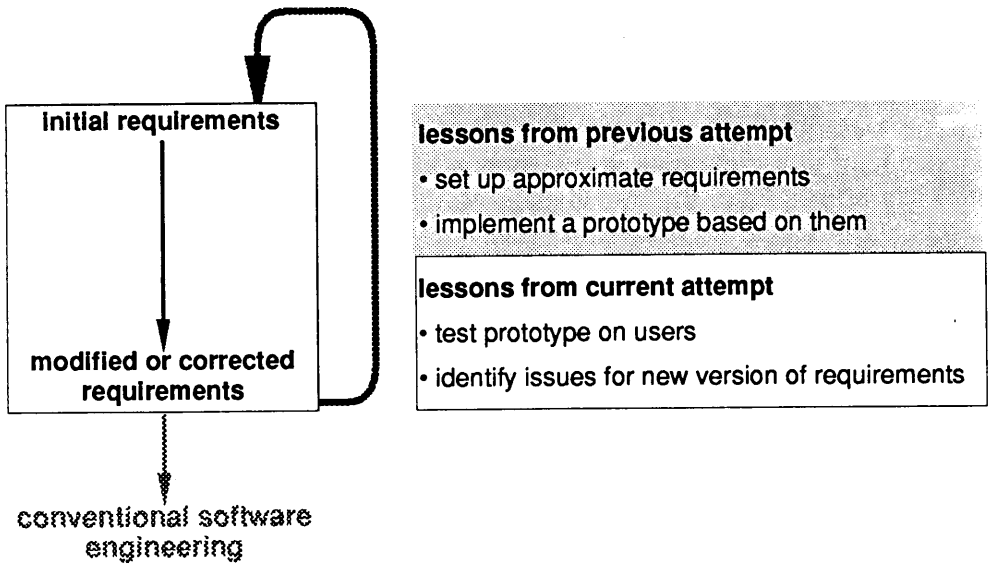
The next criterion (c2) relates specifically to reducing the complexity of the equipment. Networked or multiuser configurations will not always be complex – so exceptions should be allowed. For example, the AppleTalk network is a very cheap and simple way of sharing resources such as laser printers. However, even such a simple network can become complex to use when more sophisticated services such as file servers are added [11].

The next criterion (c3) – the software should be easy to use – requires some thought, since ease of use is not trivial to define. This point is further addressed in the rest of the paper. If the ease of use criterion can be met, the criterion of requiring minimal training (c4) should not present serious problems. Both criteria need to be specified concretely, in the context of a specific project. For example, when Apple launched the Lisa (the predecessor of the Macintosh), it was claimed that a novice could learn to use the machine in half an hour. An experiment has found that this claim is not literally true [6] – although some critics of the experiment have complained that its findings are unfair in some respects [10, 15].

The next three criteria (c5 to c7) relate to adaptability. This fact that three criteria are devoted to adaptability indicates how important this issue is. First world countries can generally rely on large markets to absorb development costs; third world countries often either have small populations or wide regional differences.

The final criterion (c8) – the cost per user should be low – also relates to spreading the usability of the software as widely as possible. A diskette costs a few cents, and hardware is becoming cheaper. Clearly, any measure of the cost per user must be heavily influenced by how widely the cost of the development of the software can be spread.

The criteria can be summarized in two major points: placing the user in control and spreading the usefulness of the software as widely as possible. Both of these points are captured (if not in full) in the concept of *usability*.



Each iteration leads to further clarification of the requirements, based on experience with a prototype for the current version of the requirements. Ultimately, should the requirements be sufficiently clear, a conventional software engineering strategy can take over. Ideally, issues elucidated should help in later projects as well.

Figure 2: Experimental software engineering

3. Experimental Software Engineering

One of the most difficult issues in software engineering is accurately capturing the user's requirements [24]. The user usually does not have a clear understanding of what a computer can do, while the software engineer may not have much knowledge about the application domain. In the context of appropriate technology, this problem is exacerbated in two significant ways.

Firstly, there is little experience in implementing software as appropriate technology. Such work as there has been has either been in the form of isolated projects [1], or of investigations into general policy issues [16]. No serious attention has been given to the problem of the issues in software engineering of meeting criteria for appropriate technology.

Secondly, potential users have had little exposure to computers, which increases the already serious difficulty of obtaining accurate requirements from the users.

In defining a strategy for dealing with these problems, two sources are drawn on in this research: other experience with software engineering in which usability has played a major role, and experience in an area in which software is created without a clear initial idea of the requirements. The example of usability is the 1984 Olympic Message System; the latter example is programming in artificial intelligence.

The Olympic Message System – which allowed athletes and other interested parties to leave messages for each other – was designed with usability as a central issue. The underlying philosophy was one of using behavioural measures to ascertain the acceptability of the system to the user. Although large numbers of people were used to test the system at later stages, the methodology was relatively informal. The strategy used deviated from the classical model of the software life cycle – sometimes known as the waterfall model [2]. The design and requirements analysis were carried through to relatively late stages. As features were implemented, they were tested on potential users for acceptability, and the system was changed as problems were identified [8]. This is in contrast to the waterfall model, in which software development is seen as consisting of largely non-overlapping stages.

The Olympic Message System is an important example, because it illustrates how far system design and construction can be driven by usability, especially in a nontrivial program. The users were from a wide range of backgrounds, and the software worked well and was accepted positively by the users [*ibid.*]. However, the potential of applying the experience of this project directly to other areas is limited in two respects. The initial functionality was reasonably clearly specified – the details were the real problem. Also, the implementers had access to a table-driven tool for generating user interfaces, which considerably increased their flexibility in changing the design. This tool was specifically intended for communications applications; such a tool may not necessarily be easy to construct for other applications.

AI is an interesting source of ideas for dealing with the issue of constructing software where the initial idea of the requirements is extremely vague. Some AI researchers view programming in AI as an experiment, in which the requirements are clarified.

r1 free switching between gathering evidence and forming, confirming and rejecting hypotheses should be supported
r2 no order should be imposed on the specific hypotheses and evidence which are considered
r3 association of hypotheses and evidence should be explicit
r4 an ordered record of the hypotheses and evidence considered should be kept
r5 the system should be non-judgmental
r6 solving of problems meaningful to the learner should be supported
r7 the student should supply medical knowledge for solving a given problem by some approximation to building an expert system – the system should not contain any medical knowledge
r8 ease of use should be emphasized; this implies that user interface may become a major issue
r9 inference should be avoided: the student, not the program, should find the solution

These requirements, which were the starting point for the experimental software engineering strategy, are based on the educational approach called knowledge engineering based learning (KEBL) as well as the criteria for appropriate technology of figure 1.

Figure 3: Initial requirements

AI is seen by some as an empirical science [17], in which programs produced by successive researchers form data points in a grand experiment [4]. The long-term outcome is an enhanced understanding of human intelligence and what computers can do. Programming in an experimental fashion requires powerful tools and techniques. Examples include the Interlisp programming environment [23] and structured growth, in which modules of a program may be rewritten as more sophisticated possibilities are considered feasible [21]. Some critics of AI have gone as far as to claim that the development of tools which support programming without a clear idea of the requirements is the major contribution AI has made [7]. Some lessons from AI research have found their way into software engineering environments, such as Cedar [22] and Pecan [20].

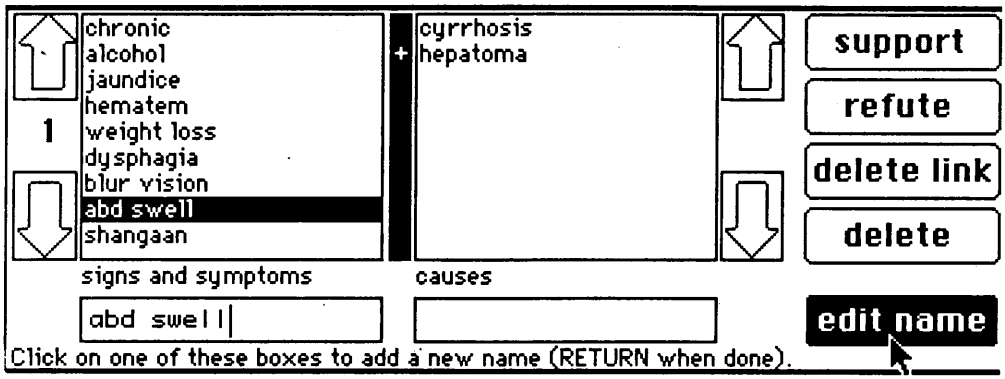
AI is however not always a suitable basis for constructing robust software, which is intended for widespread use. The lack of a precise specification of requirements makes testing difficult, and is likely to cause problems in the long term with maintenance. This claim is borne out by software engineering research which has measured the effects of replacing the requirements and design stages of the waterfall model by prototyping. The general finding is that the lack of formal documents cases problems with the later stages of the project, from integration onwards [3].

For developing software as appropriate technology, some of the ideas from the Olympic Message System and from AI can be put together. The first principle is to emphasize usability. To this end, behavioural measures of how the users relate to the software should be used. The second principle is that the requirements should be derived by a process of a succession of prototypes, each of which can be seen as a data point in an experiment to determine the correct functionality and user interface of the program. The earlier prototypes should be constructed using tools which allow maximum flexibility in changing the approach – in the AI tradition – while later prototypes should use increasingly rigorously specified languages in the spirit of software engineering. The outcome of this process, which is given the name *experimental software engineering* (ESE), is a precise specification of the requirements which may be used in a conventional software engineering project to implement the software.

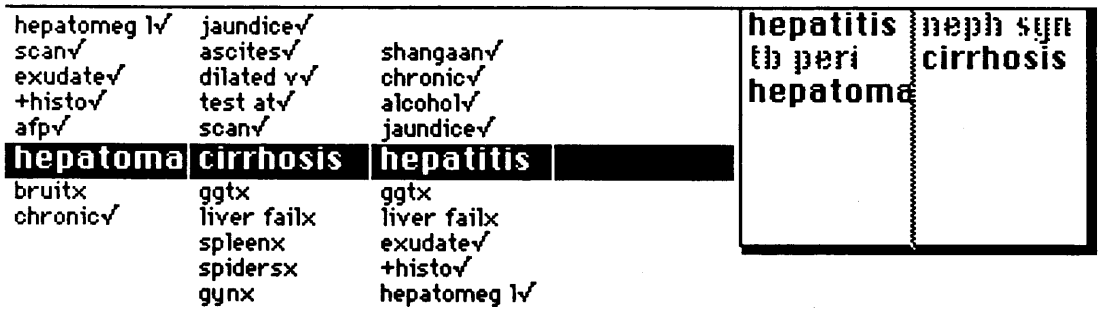
The ESE strategy is summarized in figure 2.

4. A Case Study: A Tool for Medical Education

The case study presented here illustrates how ESE may be applied in practice. The example used is the specification of requirements for a tool for medical education. An initial attempt at specifying requirements for this tool is based on the criteria for appropriate technology of figure 1. The tool is intended to be an approximation to an expert system shell, which will support learning medical problem solving in the same sense as Logo supports learning mathematical problem solving [18]. The philosophy is one of learning by doing, with the learner in control. The learner is placed in the role of a knowledge engineer, although in a much simplified version of an exercise in expert system construction. A full explanation of this step of the project is beyond the scope of this paper, which focuses on the application of ESE. This section presents a series of experi-



(a) The top half of the screen is used for adding, deleting and editing names – as well as making and breaking links. Here, a name is about to be edited.



(b) The bottom half of the screen is used for activating and deactivating causes (hypotheses), and marking signs or symptoms as present or absent. The section of the screen illustrated here contains a table of all causes on the right; those in grey are not currently active. The active ones appear in white on black writing in a report to the left of the table, with evidence for the above and evidence against below (based on the student's rules). A cause is activated or deactivated by selecting its name in the table.

Figure 4: The LISP prototype's user interface

ments which were carried out in developing the requirements, starting from the initial requirements in figure 3.

The first experiment was an initial attempt at investigating the implications of the requirements. The tool used was a program called CLASSIFY, a simple decision tree-like approximation to an expert system shell, which is supplied with Prolog-86 (an IBM PC implementation). The outcome of this experiment was used to validate the initial requirements; the requirements were then more fully tested using a program written in OPS5 on an Apple Macintosh (which is used for subsequent prototypes as well). The next version of requirements – in keeping with the philosophy of moving to tools which are increasingly useful for rigorously defining the behaviour of the program – was tested by an experiment with a prototype written in LISP. The final version of the requirements was prototyped in Pascal.

The CLASSIFY experiment mainly established communication with medical educators. A group of six people (three educators, two registrars and a medical student) which was presented with the KEBL idea was able to see merit in it, but agreed that CLASSIFY was not usable. The software proved to be limiting, and difficult to use. In particular, the fixed order in which the program asked questions based on its decision tree caused difficulty. Since this aspect of the program conflicted with the requirements, the finding was taken as confirmation that software conforming to the requirements should be constructed for another experiment.

The next prototype constructed was written in OPS5, in order to facilitate flexibility in changing the control strategy as new ideas were offered by medical educators. This prototype more fully implemented the requirements. It allowed the student to make supporting and refuting links between evidence and hypotheses, and to supply questions which the program could ask to find out if evidence was present. Once the student's rules had been constructed, the program would ask the student to give names of evidence. The student determined the order of inputs, in keeping with the learner in control philosophy. The program did not perform any inference – the idea was the student should learn to solve a problem in a systematic way by building a computer representation of the steps taken in finding the solution. The student could ask for a report of evidence for and against each hypothesis, and had to decide when to stop, as well as what the solution was. The option of switching to having the program ask the

r1 free switching between gathering evidence and forming, confirming and rejecting hypotheses should be supported – to this end, a permanent report on evidence for and against active hypotheses should be displayed, and hypotheses and evidence should be explicitly activated and deactivated
r2 no order should be imposed on the specific hypotheses and evidence which are considered
r3 rules should specify evidence as supporting or refuting hypotheses
r4 an ordered record of all actions taken by the student should be kept – including choice of hypotheses and evidence to consider, and the order of rule formation
r5 the system should be non-judgmental
r6 solving of problems meaningful to the learner should be supported
r7 the student should supply medical knowledge for solving a given problem by some approximation to building an expert system – the system should not contain any medical knowledge
r8 ease of use should be emphasized: the use of the keyboard should be eliminated as far as possible
r9 inference should be avoided: the student, not the program, should find the solution
r10 only one mode should be used for both problem-solving and knowledge acquisition
r11 signs and symptoms should be automatically considered present when brought into consideration
r12 allowance for a range of attributes in addition to *present* or *absent* should be considered

The experiments with LISP KEBL led to a further revision of the requirements, in the final application of the experimental software engineering methodology.

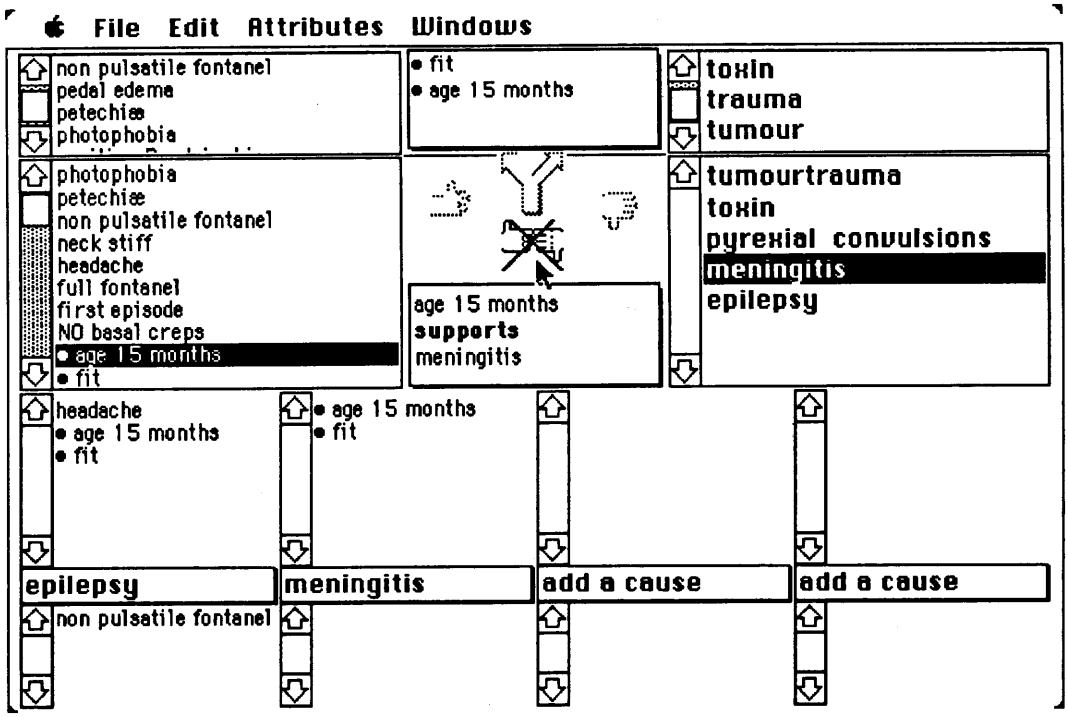
Figure 5: The final requirements

questions the student had supplied for evidence was allowed, to see whether the students would readily adopt the learner in control strategy. The program made use of the underlying LISP system's listener window to record the student's steps, and the student's rules could be saved to disk for reuse or evaluation.

The OPS5 prototype was tested with a group of 47 first-year students, who were given simple problems to solve. Informal observations were the main measure of the usability of the system. In addition, a survey of the students' attitudes was carried out at the end of the exercise, and they were given the opportunity to discuss their impressions with a group of medical educators. The survey results were generally positive; however, the survey was taken just before a long weekend, and only 51% of the students participated, so the results cannot be considered accurate. Of more significance are the observations of the students' use of the program. Typing turned out to be a major problem: the students were unable to make reasonable progress without intervention. In addition, they were only too willing to allow the computer to do the work. The "learner in control" mode of the program was generally avoided, and the mode in which the computer asked the questions was immediately used.

Based on these points – and other observations – the requirements were modified for the next prototype. The biggest innovation was the introduction of a mouse pointing device to make links; the only use of the keyboard according to these new requirements was to be in entering new names. The learner in control aspect was further emphasized, by removing the possibility of having the computer ask questions. Three other changes were matters of detail. A report indicating the evidence for and against each hypothesis under consideration was to be maintained at all times on the screen. In addition, a full record of the steps taken by the user was to be kept on disk, to allow retracing of the user's steps. This is in contrast to the OPS5 prototype, where windows on the screen needed to be explicitly saved, and the exact order of the user's steps was not recorded.

The next prototype was written in LISP. The version of LISP used allowed incremental compilation, which made for flexibility in accommodating the wishes of the three medical educators who tested early versions of the prototype. The essential functionality had ben fixed in the previous experiment, and the issue being investigated was the user interface. The LISP implementation had reasonable



The Pascal Program has only one mode, and the keyboard is not used. Names are loaded into signs and symptoms (top left) and cause (bottom right) dictionaries from disk, and are brought into consideration (into the middle part of the screen) by selecting them with the mouse. Causes are activated (i.e., become active hypotheses) by selecting their names in the "in consideration" section, and then selecting a cause position in a report (bottom). Names may be sent back to the dictionaries, and links are made or broken using the icons in the centre of the screen. Lists may be of indefinite length, and can be scrolled in the standard Macintosh style (using the mouse).

Figure 6: The Pascal prototype's user interface

access to the Macintosh graphics toolbox; other languages such as Pascal were better in this respect, but were considered less suitable for rapid prototyping. The user interface of the program is illustrated in figure 4. For this implementation, a change in terminology occurred: *signs and symptoms* replaced *evidence*, and *causes* replaced *hypotheses*. The reason for this change was to avoid a debate in medical education research as to whether medical problem solving was or was not hypothesis formation [14].

The program was initially tested in field studies at Hillbrow Hospital, Johannesburg, using a total of 14 fourth-year and sixth-year medical students. Subsequently, an experiment was conducted with 16 nursing sisters from the Soweto Community Health Centres. In both cases, findings were based on informal observations. Despite the considerably reduced reliance on the keyboard, typing remained a problem. In neither the Hillbrow nor the Soweto exercises was there time to train the users sufficiently in the use of the keyboard; all typing was done for them. In other respects, the program proved to be clumsy in detail, though the overall strategy was acceptable to the users.

It is interesting to contrast the attitudes of the medical students with those of the sisters, who had less exposure to technology. The medical students were keen on setting up a computerized guru which would find all the answers for them. One went as far as to suggest that the computer could be linked to laboratory equipment to save time in entering information. The sisters on the other hand were more sceptical. One expressed concern that using a computer in a clinical setting would cause the patient to feel neglected. Another argued strongly that the computer strategy could just as easily be carried out on paper. On the whole, though, most participants were positive and indicated interest in seeing further research.

The major weakness of the program was its use of two modes: one for making and breaking links (called *knowledge acquisition*) and one for activating and deactivating causes (called *problem solving*). Although these two modes were considered natural in terms of the original intention to emulate an expert system-building exercise, they were confusing to the users. This finding is considered important for its confirmation of earlier similar findings about the difficulty of using moded software [19]. In addition, modes can be seen as contrary to the learner in control philosophy, in that they restrict the options open to the user.

5. Evaluation of Experimental Software Engineering

The research has only gone as far as a final version of the requirements, with a matching prototype – this time written in Pascal – which, it is hypothesized, could form the basis for continuing with a conventional software engineering exercise. These final requirements are presented in figure 5; key aspects of the Pascal program are illustrated in figure 6. The Pascal program has not been fully implemented. However, the key elements of the user interface are in place, and sufficient detail has been implemented to make a detailed design reasonable straightforward.

Further research is needed into how ESE interfaces to the rest of the software life cycle. In particular, consideration should be given to including ESE tools in a software engineering environment designed to support later stages of the software life cycle. Also, work on examining the cost implications of ESE is needed. Since the ESE aims to increase the accuracy of the requirements specification, it is reasonable to suppose that the later stages – especially maintenance – should be facilitated. However, this claim needs to be justified by further research.

Nonetheless, the ESE case study has illustrated how relatively computer-naïve users can make a meaningful contribution to specifying non-trivial software, aimed at meeting criteria for appropriate technology. In addition, some issues clarified by this research are of wider application. Use of a keyboard is considered to be an inhibiting factor for users who are unfamiliar with technology. A modeless user interface, in which the user is free to choose the order of events, is found to be more intuitive than a heavily moded one. Furthermore, a learner in control strategy is found to be natural, and the learner should not have the option of allowing the computer to control the order of events. An important finding is the usefulness of AI tools on relatively cheap computers, especially as tools for ESE.

The extent to which these lessons apply generally needs further research. It would be particularly interesting to investigate whether the process of software production could become appropriate technology, so that third world countries could develop their own software industries. The case study presented here illustrates what can be achieved using relatively low-cost equipment; further research should be possible without losing sight of the intention of making the results accessible to poorer parts of the world.

Acknowledgements

I would like to thank Conrad Mueller and Prof. A.M. Starfield, who supervised the research for a Masters degree on which this paper is based, for their helpfulness and encouragement. In addition, Dr Andrew Truscott was very helpful in setting up the experiment in Soweto. Ian McNairn, Prof Graham Mitchell and Prof Pat MacPhail assisted in designing and testing the LISP prototype. The students who were prepared to be subjects of the experiments, as well as those who assisted in setting up and running the experiments, made the whole thing possible. Scott Hazelhurst assisted in proofreading this paper, and made helpful comments.

References

[1] B Auvert, P Aegerter, V Gilbos, E Benillouche, P Boutin, G Desvé, M-F Landre and D Bos, [1986], Tropicaid: Un système expert sur ordinateur portatif pour l'aide à la décision médicale dans les pays en développement, *6th International Workshop on Expert Systems and Their Applications*, 28–30, Avignon, France.

[2] B W Boehm, [1976], Software Engineering, *IEEE Transactions on Computers*, **25** (12), 1226–1241.

[3] B W Boehm, T E Gray and T Seewaldt, [1984], Prototyping Versus Specifying: A Multiproject Experiment, *IEEE Transactions on Software Engineering*, **10** (3), 290–303.

[4] B G Buchanan, [1982], New Research on Expert Systems, *Machine Intelligence* **10** (ed. J E Hayes, D Michie and Y-H Pao), Ellis Horwood, Chichester, 269–299.

[5] P Byass, [1987], Computers in Africa: Appropriate Technology? *Computer Bulletin*, **3** (2), 1987, 17.

[6] J M Carroll and S A Mazur, [1986], LisaLearning, *Computer*, **19** (11), 35–49.

[7] J Doyle, [1985], Expert Systems and the "Myth" of Symbolic Reasoning, *IEEE Transactions on Software Engineering*, **11** (11), 1361–1374.

[8] J D Gould, S J Boies, S Levy, J T Richards and J Schoonard, [1987], The 1984 Olympic Message System: A Test of Behavioral Design Principles of System Design, *CACM*, **30** (9), 758–769.

[9] N Jéquier and G Blanc, [1979], *Appropriate Technology Directory*, OECD, Paris.

[10] G Kiliany, [1987], Response to "LisaLearning" Article (letter to editor), *Computer*, **20** (3) March, 4.

[11] *MACazine*, [1987], Business Report, *MACazine*, **4** (12), 41–63.

[12] P Machanick, [1986], Low-Cost Artificial Intelligence Tools, *Quæstiones Informaticæ*, **4** (3), 27–32.

[13] P Machanick, [1988], *Design of Medical Education Software as Appropriate Technology Using Artifical Intelligence and Software Engineering* (masters dissertation), Computer Science Department Technical Report 1988–01, University of the Witwatersrand, Johannesburg.

[14] C H McGuire, [1985], Medical Problem-Solving: A Critique of the Literature, *Journal of Medical Education*, **60** (8), 587–595.
[15] D L Metzger, [1987], "LisaLearning" Called Apple-Bashing Session (letter to editor), *Computer*, **20** (3), 4.
[16] M Munashinghe, M Dow and J Fritz, [1985], *Microcomputers for Development*, CINTEC-NAS, Sri Lanka.
[17] A Newell and H A Simon, [1976], Computer Science as Empirical Enquiry: Symbols and search, *CACM* **19** (3), 113–126.
[18] S Papert, [1980], *Mindstorms*, Harvester Press, Brighton.
[19] T S Perry and P Wallich, [1985], Inside PARC: The 'Information Architects', *IEEE Spectrum*, **22** (10), 62–75.
[20] S P Reiss, [1985], PECAN: Program Development Systems that Support Multiple Views, *IEEE Transactions on Software Engineering*, **11** (3), 276–285.
[21] E Sandewall, [1978], Programming in an Interactive Environment: the 'LISP' Experience, *Computing Surveys*, **10** (1), 35–71.
[22] W Teitelman, [1984], A Tour through Cedar, *IEEE Software*, **1** (2), 44–73.
[23] W Teitelman and L Masinter, [1981], The INTERLISP Programming Environment, *Computer*, **14** (4), 25–33.
[24] P Wegner, [1984], Capital-Intensive Software Development, *IEEE Software*, **1** (3), 7–45.

This paper was received in camera-ready form.

A "Cooperating Expert's" Framework for Business Expert System Design

G R Finnie
Department of Computer Science, University of Natal, PO Box 375, Pietermaritzburg, 3200

Abstract

Expert system development environments based on current language constructs such as rules and frames have been criticised for approaching the task of problem solving from too low a level of abstraction. This paper describes the generic tasks framework for knowledge based systems proposed by Chandrasekaran et al. and discusses its application to the problem of financial statement analysis.

Keywords: *expert systems, decision support systems, financial statement analysis, generic tasks.*
Computing Review Categories: *I.2.1: Artificial Intelligence, Applications and Expert Systems*

Received October 1988, Accepted January 1989

Introduction

The development of programming languages has been categorised into a number of generations, ranging from machine code (1st) to assembler (2nd) to procedural problem solving languages like Pascal or COBOL (3rd) to non-procedural 4GLs (4th) to logic programming (5th?). Each generation step (excepting as yet the 4th to 5th) has been accompanied by a consequent leap in productivity (although some would feel that the value of 4GLs may be overstated). The expert/knowledge based system field is, in computing terms, relatively young, with problem solving being done largely in the context of the first generation of expert system tools (although again the term *second generation tools* has been used to describe the more recent development environments). Chandrasekaran and associated researchers [3,4,5,6,7] have argued that the current crop of knowledge based languages and tools operate at too low a level of abstraction for efficient problem solving, with the consequence that too much attention is focused on forcing the tools to fit the problem. They propose instead a set of tools based on the concept of generic tasks, where a specific tool is tailored to the solution of a specific class of problems.

The first section of this paper looks at some of the applications of knowledge based systems in business, and more specifically in the area of finance. This is followed by a section which outlines the generic tasks framework and some of the tools available. The next section discusses the problem of financial statement analysis by expert systems and suggests how the generic tasks approach could be used in its solution. The conclusion briefly considers the potential advantages of approaching expert systems development from a higher level of abstraction than the current tools provide.

Business Knowledge-Based Systems

In view of the high potential payoff of expert (knowledge-based) systems in certain commercial applications, their uses or possible uses in business have aroused wide interest. Several texts have been published in the area [e.g. 11,14,24] and a number of conferences or conference sessions have concentrated on the topic [e.g. 20,26]. It is apparent from the literature that there has been some success in developing useful commercial expert systems. However, some authors have questioned the value of expert systems in this area. Martins [17], for example, argues that they have proven effective only in simple applications, are weak theoretically, have high development costs and require heavy utilisation of resources.

The relationship between expert systems (ES) and decision support systems (DSS) has also been the subject of some discussion. Turban and Watkins [28] suggest that there is a strong case for treating ES as a component or subsystem of a DSS. They argue that although there is some support for the view that ES are a special case of DSS, there are several fundamental differences separating the two areas, in particular the fact that in many cases ES make rather than support decisions. A similar view is held by Pfeifer and Luthi [23] who hold that the two concepts should be treated as complementary rather than similar. They state "the paradigm for DSS is improving (management) decision making, the one of ES problem solving". Sen and Biswas [25] argue that the ES approach could be used to establish domain independence in DSS (called XDSS) and to simplify the integration of qualitative and quantitative data. As with similar debates on nomenclature (e.g. DSS vs MIS), it is unlikely that a universally acceptable definition will emerge.

A broad range of potential commercial application areas for expert systems have been identified in the research literature. Blanning [1] has suggested four primary areas for the use of expert systems in management. These are (1) resource allocation: allocating limited resources under various constraints to several activities, (2) problem diagnosis: using stored knowledge to explain symptoms associated with problems, (3) scheduling and assignment: also under various constraints, and (4) information management: assisting managers to assemble information required to solve specific problems. Rauch-Hindin [24] identified several current uses of ES is the business and finance area. These included financial statement analysis, mergers and acquisitions, portfolio analysis, asset and liability management, expert databases and intelligent reporting systems. Some other applications which have been investigated are insurance risk assessment, tax accounting, investment planning, auditing, banking services advice, strategic decision making and the analysis of competition for marketing tasks.

The finance area has attracted particular attention for expert system development, possibly in view of the high potential return in certain applications as well as a lack of personnel with sufficiently high skills in areas like investment advice. Rauch-Hindin [24] observes however that the financial services market may develop more slowly than the industrial market, as the sector may not lend itself to incremental systems development. Much of the work in this area has been done in secret [24] in order to provide financial advantage to the developing company but a number of authors have published on their research. Mays et al. [18] describe a system based on semantic nets which establishes ways of financing large equipment proposals. Dhar and Croker [9] have considered the use of ES techniques to simplify the use of integer programming models in resource allocation problems. Methlie [19] used a rule based system built on EMYCIN to study ratio analysis in financial statement analysis. Mui and McCarthy [22] describe the design of a financial statement analyser which can use semantic nets to aid the interpretation of additional financial statement information such as footnotes. Kerschberg and Dickinson [15] developed a PC based *expert support system* for financial analysis which incorporated the concept of *intelligent spreadsheets* i.e. linking a spreadsheet with a knowledge engine. Heuer et al. [13] describe a frame-based system to provide investment advice. Duda et al. [10] have applied a functional language approach to develop a system for assessing the risk associated with business opportunities. Stansfield and Greenfield [27] have developed Planpower which generates comprehensive personal financial plans. This list is by no means exhaustive and there remain a considerable variety of financial applications for knowledge based systems which have not yet been researched or reported.

The Generic Tasks Approach to ES Development

A central theme of the research performed at the Laboratory for Artificial Intelligence Research (LAIR) at Ohio State University has been the identification of several *generic* tasks which provide the basic framework for the solution of different classes of expert system problem [3,4,5,6,7,8,12]. Chandrasekaran [7] argues that the current set of ES problem solving tools (rules, frames, semantic nets, logic programming, etc.) provide too low a level of abstraction to adequately reflect the structure of the problem under consideration. He states [7] "The available paradigms often force us to fit the problem to the tools rather than fashion the tools to reflect the structure of the problem". More *conventional* programming languages have moved through several levels of abstraction from the second generation of assembler, to the third generation problem solving languages (e.g. Fortran, Pascal) to the current 4GL tools. Chandrasekaran and his research group hold the view that most currently available AI languages are in effect the assembly languages of the field and that higher level languages and constructs are required to facilitate effective ES development.

Bylander and Chandrasekaran [5] characterise a generic task in terms of the following information:

(a) The type of problem i.e. what function does the generic task perform
(b) The representation of knowledge i.e. how should knowledge be structured to facilitate solution of the generic task.
(c) The inference strategy i.e. what inference techniques should be applied to the knowledge representation.

The research has identified six generic tasks which have been developed for the construction of expert systems. The following is a brief review of the salient features of each type of task, more detail can be found in [7]. The first four tasks arose largely from work on diagnostic reasoning while the last two came from studies of routine design problems. Routine design can be applied when the way to decompose a design problem is already known.

(1) Hierarchical Classification

The diagnostic process can be viewed largely as one of classification driven by the symptoms or known facts concerning the specific problem. In the hierarchical approach, each node in the classification hierarchy corresponds to some diagnostic hypothesis. Problem-solving proceeds in a top-down manner: the root node (highest level concept) is given initial control of the search, control then passes to the *most likely* successor node, and so on. More general concepts appear in the higher nodes of the structure while more specific concepts are placed lower in the hierarchy. Each node or concept in the hierarchy contains sufficient knowledge to enable it to decide

how well this concept matches the current data. If a concept is insufficiently supported i.e. assumed not relevant to the problem, all its successors are also assumed irrelevant. The terminology adopted by the research group is that each node plays the role of a *specialist* i.e. the conceptual hierarchy is a community of specialists [7]. A blackboard system is used to provide interaction between the specialists and to maintain a record of the current state of the system.

The inference strategy employed is termed **establish-refine**. Each node attempts to establish itself. If it fails, the search structure is pruned at this point and the next node at the same level is considered. If it succeeds in establishing itself, the refinement process is started i.e. each successor attempts to establish itself. Gomez and Chandrasekaran [12] state that three types of rule can be used in each concept or node: (a) confirmatory rules which look for evidence associated with the concept, (b) exclusionary rules which can be used to eliminate the concept, (c) recommendation rules which are pieces of knowledge which suggest direction to the sub-concepts. The process continues until sufficient tip nodes (corresponding to *primitive concepts* or specific hypotheses) have been established to account for all the observations or symptoms.

(2) Hypothesis matching or assessment

During the classification process, each node or specialist must attempt to establish itself, usually under conditions of considerable uncertainty. The essential process involved is that of pattern matching of the concept against relevant data to establish a measure of *goodness of fit*. Chandrasekaran [7] argues that the matching process should be treated as a separate generic task with separate knowledge structures and inference strategy. The matcher developed by the research group for the MDX diagnostic system [12] employed a process of hierarchical symbolic abstraction. An abstraction of the data is mapped to a set of discrete qualitative measures of fit – the process is hierarchical as the final abstraction is computed from intermediate abstractions. More detail on the process is available in [4]. Alternative matching processes could be considered – the primary issue under consideration is that the problem of matching hypotheses and data can be considered as a general type of problem solving process which could be employed in a variety of contexts.

(3) Knowledge-directed information passing

A problem solving process does not operate in isolation but instead functions in the context of general knowledge of the problem domain. For example, the analysis of financial data should be considered not only for a particular set of financial statements for a particular company but also within the context of the economic environment, general principles of accounting reporting and general knowledge concerning financial issues. Inferences concerning this type of knowledge can be handled by a generic task called knowledge-directed information passing. Mittal [21] developed a specific database system for the task (PATREC) which can hold general domain knowledge together with an inference process which can be used to infer domain knowledge which is not explicitly stored in the database. The PATREC system is organized as a frame hierarchy, each frame containing either the required data or information about how the value might be obtained. Again the specific storage design and inference mechanism is not as significant as recognising the need for a specific generic task to handle this type of problem-solving.

(4) Abductive assembly

Although hierarchical classification may generate a number of hypotheses, each hypothesis tends to be developed and viewed in at least partial isolation from the others. The Ohio State Laboratory for Artificial Intelligence has proposed a process of abductive reasoning which can be applied to a set of tentative hypotheses to build a composite hypothesis which best explains the data. Each hypothesis has some set of symptoms which it can account for with some degree of certainty. The abductive assembly process develops the composite hypothesis incrementally by successively adding hypotheses until a *best* or *most economical* coverage of the total problem is obtained.

The abductive assembly process requires knowledge in the form of relations between the relevant data and the hypotheses e.g. relations such as incompatibility between the data and a hypothesis or the data being suggestive of a particular hypothesis. The inference process can be described as alternating assembly and criticism. During assembly, the search process has the goal of explaining all significant data and is driven by a means-ends heuristic strategy. Each assembly seeks to add the best hypothesis which explains the most significant portion of the remaining data to the composite hypothesis. After each assembly, the criticism phase removes any superfluous components of the hypothesis. The process continues until all the data is explained or there are no hypotheses left.

(5) Hierarchical design by plan selection and refinement

In routine design, compiled design plans are assumed to be available for each stage in design [3]. Knowledge for this generic task should be considered in two forms: (1) knowledge of the structure of the objects which are known at some level of abstraction i.e. the device components and their relationship to each other, and (2) knowledge of the design plans for each part of the structure being developed. The plans

can make design choices and invoke subcomponent designs. In the Aircyl system [3], the knowledge is organised as a design specialist hierarchy. Aircyl is used for the design of air cylinders, which can be considered a routine design problem as the general structure of air cylinders is known as are the design plans for each air cylinder component.

The control process works recursively to establish a complete design by initially selecting a specialist corresponding to an object component. The specialist selects a specific plan which may in turn suggest the activation of other specialists. If a plan fails, the reasons for the failure may cause a higher level specialist to change the plan and retry the problem.

(6) State abstraction

In many practical problems, an expert system user is interested in predicting the consequences of actions i.e. the well-known *what-if* capability of decision support systems. Typically, solution of this type of problem requires some form of simulation.In simplified form, the knowledge structure should be capable of representing the relationship between any change of state in any subsystem and the consequent state change in the system immediately containing the subsystem. These relationships could be represented as a hierarchy of *conceptual specialists* [7]. The control process operates in a bottom-up manner by tracing the effect of a specific state change through the various affected subsystems until the effect of the change can be determined at the required level of interest.

A variety of languages and tools have been developed by the Ohio State University Laboratory for Artificial Intelligence Research group to support the encoding of generic tasks. The process of hierarchical classification has resulted in the development of the language CSRL (Conceptual Structures Representation Language) [6]. The language HYPER [4] uses structured matching to measure the fit of a hypothesis to a problem situation. DSPL [3] was developed to cater for the process of object synthesis by selection and refinement. The knowledge-directed database system PATREC [21] has been used for knowledge-directed information passing while a language for abductive assembly (PEIRCE) is under development. These languages could considerably simplify the encoding of expert system tasks by facilitating problem definition and programmed solution at the correct level of abstraction.

An Example From Financial Statements Analysis

Methlie [19] has observed that the financial position of a company has much in common with the health of a patient where the financial statements can be considered as indicators of the financial health of the firm. However, as in medical diagnosis, the analysis and interpretation of the financial statements is a non-trivial task. The information must be consolidated and organised into a compact understandable form. A primary tool for this organisation is ratio analysis, in which the ratios between specific financial values can be used as indicators of possible malfunction in a given sector of the organisation. An example is the current ratio which is simply the ratio of current assets to current liabilities. A poor current ratio could indicate that either current assets or current liabilities are unsatisfactory. Further investigation might reveal low levels of assets like accounts receivable, inventory and cash or high levels of various types of debt.Six groups of financial ratios may be used in analysing the financial state of an organisation [29]:

(a) Liquidity ratios are a measure of the firms ability to meet its short-term commitments from liquid assets.
(b) Leverage ratios are used to indicate the capacity to service long term debt.
(c) Activity ratios illustrate how assets are being utilised.
(d) Profitability ratios indicate net return on sales and assets.
(e) Growth ratios which assess the firm's ability to maintain its economic position.
(f) Valuation ratios which assist in focussing on the goal of maximising the value of the firm.

If the simple determination of ratios was sufficient to determine the financial state of a company, there would be little need for knowledge based systems. However, the analysis is complicated by several issues. Ratios should not be considered in isolation but should be studied in the context of factors like industry averages, the general state of the economy and demographic considerations. This type of comparison requires considerable skill and general financial knowledge e.g. the knowledge that a particular industrial sector is under stress. Much of this type of data is qualitative rather than quantitative. Factors such as the relative size of the company should also be considered. Ratios should not be treated individually but should be considered together i.e. certain ratios may reinforce or negate decisions based on other ratios. Another complicating factor is that ratios should be analysed over time i.e. trends and discontinuities should be noted and related to such issues as past business decisions.

However, probably the most complex problem for a knowledge based system to handle is that the meaning of the financial data can vary according to the accounting principles and explanatory statements included in the financial data. Rauch-Hindin [24] points out that the relatively rare skill of a successful financial statement analyst is "generally characterised by their ability to read a financial statement's footnotes" where footnotes are defined as

explanations of the criteria used to prepare the financial statement. Interpretation of the footnotes establishes the financial environment in which the analysis should proceed. The FSA system of Mui and McCarthy [22] is intended to cater for the preliminary data gathering or intelligence phase of a financial analysis (called the familiarisation phase). A major component of this system, a frame based representation of footnotes, is responsible for interpretation of the footnotes to establish the *general financial knowledge* required for effective analysis of the financial data. The FINEX system of Kerschberg and Dickinson [15] however ignores this issue of interpreting the additional financial statement data and assumes that spreadsheet models of the statements can provide sufficiently accurate ratios for analysis. This system is however interesting for its use of a semantic net to define the relationships between various financial concepts and the ratios. The semantic net also provides a control strategy for eliciting data by questioning the users.

The view that a company financial position and a patient's health are analogous suggests that a knowledge based system for the analysis of financial statements could be similar in structure to those used in medical diagnosis. Mui and McCarthy use a framework for financial statement analysis based on a protocol analysis performed by Bouwman [2] which divides the decision process into the phases of familiarising and reasoning. Familiarisation involves data gathering and searching the environment to identify key values in the statements and potential problem areas. Successful familiarisation requires interpretation of footnote data. The reasoning phase utilises not only the ratios but also requires the type of knowledge discussed above e.g. knowledge of the economy and the industry.

The generic tasks approach suggest a possible system design framework for financial statements analysis (FSA) which allows the process to be subdivided into a number of co-operating experts, each expert comprising a specific generic task. The initial phase of financial statement analysis could be considered a form of diagnosis in which the system would be attempting to ascertain whether anything is wrong in any part of the company. This could be approached using the generic task of hierarchical classification. A partial classification hierarchy is outlined in Figure 1 with the diagnostic knowledge of FSA distributed throughout the hierarchy. The establish-refine problem-solving paradigm could be applied to drive the classification process. The ratios node would attempt to establish if a financial problem exists, if so, the refinement would consist of attempting to establish each successor i.e. to determine whether the problem lies in short-term obligations, long term debt requirements, asset use, profitability or growth (or some combination of these). Each of these concepts would then attempt to establish itself before refining its successors. For example, if a problem is suspected in long-term debt servicing, the focus would the shift to attempting to establish either a balance sheet or income statement class of problem (or combination). The hierarchical classification technique provides a suitable control regime for effective data gathering i.e. the data gathering is performed at each point of diagnostic focus.

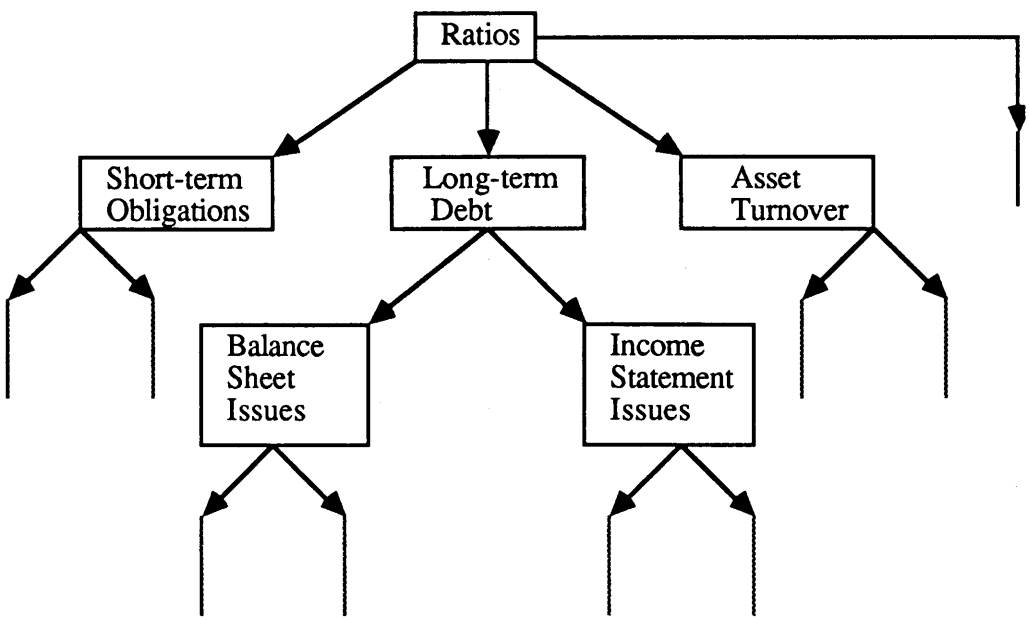


Figure 1 Hierarchical Classification for Financial Statement Analysis

The process of establishing a specific concept (node in the hierarchy) can be approached using the generic hypothesis matching task. The technique can employ various forms of uncertainty, one possible approach being the qualitative technique used by the HYPER system [4]. The task essentially involves matching a concept to specific data to establish a *goodness of fit* measure. The matching process concentrates on the accumulation of evidence which either supports or rules out a specific diagnosis with some qualitative or quantitative measure of confidence. For example, in establishing short term liquidity problems, attention would be focussed on determining indicators such as the current ratio and the acid test ratio. If the focus was on long term income statement problems, evidence collection would be aimed at data like fixed charge coverage and cash flow coverage.

Hierarchical classification and hypothesis matching require the interpretation of financial statement inserts and footnotes as well as general data on the industry and the economy. The process of knowledge directed data passing could be used for the intelligent retrieval of this data. The frame based approach used in PATREC can be used to hold general domain knowledge with each slot in a frame containing either the required knowledge or information about how to determine the required knowledge. Procedural attachment could be used to generalise the knowledge gathering process. Mittal et al. [21] also discuss techniques for the representation and processing of temporal knowledge which could be of value, inter alia, in the processing of trend information. Within the context of financial statement analysis, the classification and matching processes could be investigating the hypothesis of problems in the long-term debt structure of the company. However, the effect of leverage depends, inter alia, on the state of the economy and the degree of risk averseness of the investors. When the economy is in a down cycle, firms with low leverage ratios have a lower risk of loss but they also have lower probable returns if the economy enters an up-swing. The reverse is true for firms with high leverage. Knowledge concerning the relationship between leverage, as measured by the ratios, and factors such as the economy or the risk profile could be held in a frame-structured database for retrieval as required.

The classification process, in concert with matching and information passing, could produce a number of possible hypotheses to account for the firms financial state. The hypotheses would in most cases not be independent. The abductive assembly process could be used to assemble a composite hypothesis which best explains the data i.e. a hypothesis which provides a minimal covering set to explain all the financial symptoms. As a simple example, to explain a low profit margin on sales, the hypothesis of high costs could be supported (rather than that of low sales prices). To explain high interest expenses, there could be a supported hypothesis of excessive investment in plant and equipment. However, the cause of the high input costs and hence the low profit margin could be the high interest charges.

Conclusion

The generic tasks approach appears to have some potential for the efficient development of business knowledge based systems. Many business problems do not readily lend themselves to expert system design based on existing tools, which tend to provide a relatively uniform and low level set of representations. The higher level of abstraction proposed by Chandrasekaran et al could provide the ability to attack larger problems by providing a framework for problem decomposition as well tailoring the method to the problem. An example could be the area of strategic planning, in which a wide variety of qualitative and quantitative sources of data and techniques are needed. This group also argue that the generic tasks approach assists in providing a clearer explanation of the problem solving employed than the current tools, a critical component of any expert system. Bylander and Chandrasekaran [5] also claim that the approach will simplify knowledge acquisition by suggesting a different knowledge acquisition strategy for each kind of reasoning.

If the current set of knowledge base tools can be considered the equivalent of assembler level programming for expert systems, dealing with the problem solving at a higher level of abstraction has the potential of providing the same leap forward in knowledge base computing that languages like Fortran and COBOL provided over assembler coding. The building of sophisticated expert systems is currently a very slow process, even with development environments, and any advances which could dramatically improve the rate of development are worth serious investigation.

References

[1] R W Blanning, [1985], Expert Systems for Management: Research and Applications, *Journal of Information Science*, 9, 153-162.

[2] M J Bouwman, [1983], Human Diagnostic Reasoning By Computer: An illustration From Financial Analysis, *Management Science*, 29, 653-672.

[3] D C Brown and B Chandrasekaran, [1986], Knowledge and Control for a Mechanical Design Expert System, *IEEE Computer*, **19** (7), 92-100.

[4] T Bylander and T Johnson, [1988], Structured Matching, Tech Report, The Lab for AI Research, Ohio State University, Columbus, Ohio.

[5] T Bylander and B Chandrasekaran, [1987], Generic tasks for Knowledge-Based Reasoning: the "Right" Level of Abstraction for Knowledge Acquisition, *Int J Man-Machine Studies*, 26, 231-243.
[6] T Bylander and S Mittal, [1986], CSRL: A Language For Classificatory Problem Solving and Uncertainty Handling, *AI Magazine*, **7** (3), 66-77.
[7] B Chandrasekaran, [1986], Generic Tasks In Knowledge-Based Reasoning: High-Level Building Blocks for Expert System System Design, *IEEE Expert*, **1** (3), 23-30.
[8] B Chandrasekaran and M Tanner, [1986], Uncertainty handling in Expert Systems: Uniform vs Task-Specific Formalisms, in L Kanal and J Lemmer (eds), *Uncertainty in Artificial Intelligence*, Elsevier Science Publishers, 35-46.
[9] V Dhar and A Croker, [1988], Knowledge-Based Decision Support in Business: Issues and a Solution, *IEEE Expert*, 3, 53-62.
[10] R O Duda, P E Hart, R Reboh, J Reiter and T Risch, [1987], SYNTEL: Using a Functional Language for Financial Risk Assessment, *IEEE Expert*, **2** (3), 18-31.
[11] C J Ernst, (ed), [1988], *Management Expert Systems*, Addison-Wesley Publishing Company.
[12] F Gomez and B Chandrasekaran, [1984], Knowledge Organization and Distribution for Medical Diagnosis, in Clancey and Shortliffe (eds), *Readings in Medical Artificial Intelligence*, Addison-Wesley Publishing Company.
[13] S Heuer, U Koch and C Cryer, [1988], Invest: An Expert System for Financial Investments, *IEEE Expert*, **3** (2), 60-68.
[14] C W Holsapple and A B Whinston, [1987], *Business Expert Systems*, Richard D Irwin Inc.
[15] L Kerschberg and J Dickinson, [1988], FINEX: a PC-based Expert Support System for Financial Analysis, in C J Ernst (ed), *Management Expert Systems*, Addison-Wesley Publishing Company, 111-133.
[16] P Levine, J Ch Maillard and J Ch Pomerol, [1987], DECIDEX, a Multi-expert System for Strategic Decisions, in H G Sol, C A Takkenberg and P F de Vries Robbe (eds), *Expert Systems and Artificial Intelligence in Decision Support Systems*, D Reidel Publishing Company.
[17] G R Martins, [1984], The Overselling of Expert Systems, *Datamation*, **3** (18), 76-80.
[18] E Mays, C Apte, J Griesmer and J Kastner, [1987], Organizing Knowledge in a Complex Financial Domain, *IEEE Expert*, **2** (3), 61-70.
[19] L B Methlie, [1987], On Knowledge-Based Decision Support Systems For Financial Diagnosis, in C W Holsapple and A B Whinston, (eds), *Decision Support Systems: Theory and Applications*, Springer-Verlag, 335-371.
[20] G Mitra, [1986], *Computer Assisted Decision Making*, North Holland.
[21] S Mittal, B Chandrasekaran and J Sticklen, [1984], PATREC: A Knowledge-Directed Database For a Diagnostic Expert System, *IEEE Computer*, **1** (9), 51-58.
[22] C Mui and W E McCarthy, [1987], FSA: Applying AI techniques to the Familiarization Phase of Financial Decision Making, *IEEE Expert*, **2** (3), 33-41.
[23] R Pfeifer and H-J Luthi, [1987], Decision Support Systems and Expert Systems: a Complementary Relationship, in H G Sol, C A Takkenberg and P F de Vries Robbe (eds), *Expert Systems and Artificial Intelligence in Decision Support Systems*, D Reidel Publishing Company.
[24] W B Rauch-Hindin, [1985], *Artificial Intelligence Applications in Business, Science, and Industry* vol 2: applications, Prentice-Hall.
[25] A Sen and G Biswas, [1985], Decision Support Systems: An Expert Systems Approach, *Decision Support Systems*, 1, 197-204.
[26] H G Sol, C A Th Takkenberg and P De V Robbe, [1987], *Expert Systems and Artificial Intelligence in Decision Support Systems*, D Reidel Publishing Company.
[27] J L Stansfield and N R Greenfield, [1987], Planpower: A Comprehensive Financial Planner, *IEEE Expert*, **2** (3), 51-60.
[28] E Turban and P R Watkins, [1986], Integrating Expert Systems and Decision Support Systems, in R H Sprague and H J Watson (eds), *Decision support Systems: Putting Theory into Practice*, Prentice-Hall.
[29] J F Weston and E F Brigham, [1981], *Managerial Finance*, (7th ed), The Dryden Press.

The Application of Scientific Method to Information Systems Analysis

P M Q Lay and C R Atkinson
Department of Accounting, University of Cape Town, Private Bag, Rondebosch 7700

Abstract

This paper addresses the challenging question of the rigour of the systems analysis process. First of all it discusses the nature of analysis, concluding that, contradictory to current opinion, it can best be described as an art. The processes of research and systems analysis are then mapped onto each other and the similarities are described. Some of the methods used by scientific researchers are then analyzed and a framework is proposed whereby the techniques and procedures for research can be integrated into the systems analysis process, thereby improving its overall rigour.

Received June 1988, Accepted March 1989

1. Introduction

Systems analysis, as a discipline, is approximately forty years old. While it is recognized as a vital task in the building of information systems, the principles upon which it is based, and the underlying philosophies, are in their infancy. An analysis of the literature dealing with systems analysis reveals that it has moved through three distinct stages. The early writings on the topic were fundamentally experiential with authors emphasizing practical methodologies. In the sixties there was a swing towards building a more rigorous approach as authors delved into systems theory and the scientific method to build a stronger foundation. In the third (and current) stage authors have reverted to the practical era, and this despite the emergence of MIS as an academic discipline [5]. Prominent researchers such as Couger and Knapp [4] state categorically that systems analysis has moved from the realm of an art to that of a science. In direct contrast to this the conclusion in this article is that because of the non-repeatability of the solutions and the objects of the analysis, the total analytical function can at best be described as an art. However, this should not be used as an excuse to down-grade the process to the level of witchcraft. Within the individual activities that comprise systems analysis there is scope for the injection of scientific procedures, provided that the philosophy and objectives are clearly understood.

The aim of this paper, therefore, is to revitalize the scientific approach to systems analysis and to draw comparisons between the processes of research and systems analysis, and to provide a basis on which to improve the analytical process. To achieve this end, three main areas will be examined. First, aspects of systems analysis and positivist science will be discussed. Scientific research will then be described and the elements of rigorous research will be superimposed on the analytical process. Suggestions will then be made relating to specific aspects of the information systems analysis phase of systems development that would improve the rigour of the process.

2. The Scientific Method

The scientific view of the world is basically of a system in which all processes are governed by unchanging laws, and the aim of the pure scientific endeavour is to advance knowledge about those laws and processes - what they are and why they are. In pursuit of this aim, modern science employs three principles: reductionism, repeatability of experimentation, and refutation. Reductionism, which is used to simplify problems, has its foundations in Descartes' view of a problem in terms of parts or levels:

> ".... to divide each of the difficulties that I was examining into as many parts as might be possible and necessary in order best to solve it." [3]

The second principle of scientific method, repeatability of experimentation, distinguishes it from other disciplines. Proof of a theory is not dependent upon rational argument of why it should be correct, but on physical demonstration. Checkland [3] speaks of science as public knowledge which is demonstrated through experimental happenings. It is a knowledge which is not affected by interpretation, emotions, human bias or irrationality. Connected with the repeatability of experimentation is the measurement of values obtained. Facts expressed as quantitative results have more standing than qualitative findings, and are also more easily repeatable.

The third characteristic, that of refutation, is achieved through sequences of experiments which are used to test hypotheses. Each experiment re-tests and builds on the previous, thus adding to

knowledge (hence this process is also referred to as cumulative progress). The lack of cumulative progress in the information systems field has been written about by prominent IS researchers [5,11].

3. Views of Systems Analysis

A comprehensive scan of the literature on systems analysis indicates that the systems analysis function has moved through three discrete stages. Early writers regarded it as an art form, with the analysis aimed at documenting existing processes only [4]. In the mid-sixties this view changed as authors and researchers looked for a more rigorous approach. Lee, in one of the definitive publications on systems analysis [13] constructed an approach based on the scientific method. He wrote of problem definition (specification of the boundaries of the system), research (an observation of the operation of the systems and a specification of the problems), and the development of models coupled with the formulation of hypotheses. This built on the work of Johnson, Kast and Rosenzweig [12] who refer to the scientific method as a basis for systems analysis. This attempt at a rigorous approach to systems analysis was pursued by others such as Hare [8] who, as early as 1967, advocated a prototyping approach and the use of techniques such as "symptom-cause complex tables" and "differential diagnosis" as methods of understanding relationships and activities within a system. All these efforts were aimed at placing systems analysis on a firmer footing than the art form suggested by earlier authors.

In the third phase there seemed to be a reversion to the earlier approach. For instance, an analysis of books by Hodge and Clements [9], Licker [15], Leslie [16] and Davis [7] to name but a few, reveals almost no reference to the scientific approach with its emphasis on repeatability and rigour. This would force the conclusion that either the original suppositions were false, and that the scientific method had no place in systems analysis, or that authors were ignorant of the work of their predecessors. The latter being the case, MIS researchers are guilty of trying to build the systems analysis foundations in a vacuum instead of building on the works of others - the basis for good research and knowledge development.

4. Positivist Science and Systems Analysis

There are points on which the approaches of positivist science and systems analysis agree, but they vary greatly on the fundamental issues of methods of observation, confirmation of results and repeatability. Systems analysis is largely based upon subjective enquiry, due to the fact that the systems investigated are of human creation, whereas scientific enquiry must be neutral. Another significant difference between the two can be found in their fundamental aims: the aim of science is the advancement of knowledge (Aristotle's theoria) whereas the primary aim of information systems analysis is to improve organizational work or data flows or to provide data to improve or aid decision making. It is therefore fairly obvious that the two are not compatible on major issues and that if the *positivist* definition of science is accepted, information systems analysis cannot be regarded as a science.

Does this mean that systems analysis is therefore relegated to the area of "black (or grey) art" or witchcraft (with the association of unstructured, ill-defined paths)? This would seem an easy solution. A more constructive approach would be to accept the artistic domain but to attempt to superimpose some of the disciplines and techniques of scientific enquiry into the procedure of systems analysis - not with the objective of lifting it into the realms of science but simply of improving its rigour and repeatability and therefore the correctness of the results.

This paper therefore proceeds by investigating the philosophy of research and its relationship to systems analysis. If the relationship holds then aspects of research (or scientific enquiry) could be transferred to analysis and built into the analysis techniques.

5. Systems Analysis and Research

Research is the process of careful search, of systematic investigation. Although there are many forms that research can take, the process aims at understanding a specific problem and building a solution. A research project, according to Howard and Sharp [10] consists of four phases: planning, data collection, analysis of the data and the formulation of conclusions, and the presentation of results. Leedy [14] refers to a cycle of planning, problem analysis, data collection and analysis and result presentation. Now compare this to the systems analysis process. Ahituv and Neumann [1] describe the analysis of a system as three stages: the preliminary analysis - during which the problem is analyzed and data is collected, the feasibility study when conceptual solutions are designed, and information analysis when the final solution is developed and the detailed systems proposal presented. These stages are referred to

by many other authors (eg Davis and Olsen [6], Lucas [17]) and demonstrate that at the macro level there is a close correlation between the processes of research and systems analysis. They both aim at the analysis of a problem and the creation of an appropriate solution (by adapting or inventing). They both aim at documenting the results so as to convey those results to a third party.

If the above argument is accepted then it follows that the systems analyst is a form of researcher and that elements of rigorous procedure should be as applicable to the one as to the other. Since the process of research is far older than that of systems analysis this paper focuses on that aspect - the procedures advocated by prominent philosophers of science.

The **Baconian** method advocates that there is no formal hypothesis, that the researcher should proceed to immediate empirical observation and deduce conclusions based on observations. His rigour is obtained through the nature of his observation. This inductive approach has received significant criticism in recent times by people such as Sir Peter Medawar [18] and Karl Popper [20]. Note, however, that it was fashionable in DP circles a number of years ago to refer to the unbiased nature of the analysis phase and to suggest that the analyst should refrain from formulating a solution (pre-judgement) until he was aware of all the facts (advocating the Baconian approach). This argument (thankfully) has been contested by a number of authors and the studies of Vitalari and Dickson [22] appear to support the contention that not only are hypotheses natural but that good analysts exploit them. An hypothesis can be liberating, leaving the researcher free to let his/her thoughts roam while proving the hypothesis.

Cohen and Nagel were at the other end of the extreme, advocating immediate hypotheses which, they claimed, were research directing. This is followed by inductive reasoning and observation. The problem with this approach is that the researcher formulates an hypothesis when there is insufficient data - much like an analyst suggesting a solution after a user telephones and says: "There is a problem with my system". The danger is that the analyst then protects his hypothesis and makes the facts fit the solution rather than vice versa.

The compromise approach, and the one that seems to have achieved the greatest acceptance by twentieth century researchers, was proposed by Galileo [19]. This has come to be known as the hypothetico-deductive approach and consists of the following steps:

Problem reduction to root forms
Selection of the simplest phenomena relating to the problem
Empirical observation of these phenomena
Projection of hypotheses
Deduction based on the hypotheses
Empirical proof of the hypotheses (relating the problem to the hypotheses)
Generalization

The correctness of each stage is established before proceeding with the next.

There can be no doubt that any researcher is measured in terms of not only the nature and extent of the project (the relevance to the object system), but, as importantly, the rigorous proof of his hypotheses. Failures of research are usually attributed either to an incorrectly specified hypothesis or to this lack of rigour. The many tools and techniques that are available to the researcher aim at improving his ability to understand a problem and to prove the results of his research. The tools and techniques available to the analyst, on the other hand, may help him to describe the object system in terms of flows but do not help him validate hypotheses concerning the true nature of the problems or solutions, ie they are description-orientated, not solution-orientated.

6. Applying a Research Methodology to System Analysis

Assuming that the approach of Galileo is the most rigorous and relevant to systems analysis, how could it be adapted to fit the systems analysis process? A possible methodological statement would consist of the steps shown in Figure 1. The point to note about the sequence is the emphasis on defining and understanding the problem and the focus on proof. The statement of an hypothesis and formulation of a solution is relatively late in the cycle.

7. On the Use of Hypotheses

The hypothesis is probably the most important intellectual instrument in research. Its function is to give direction in experimentation and observation, leading to discoveries even when not correct itself [2]. Analysts should beware of protecting the hypothesis through manipulation of the experimental results. Ideas should be subordinated to the facts, not vice versa. This can, to a certain extent, be overcome by the formulation of multiple, but not conflicting, hypotheses.

SCIENTIFIC METHOD	USE IN SYSTEMS ANALYSIS
Understanding the problem domain.	The boundaries and scope of the problem are defined.
Stating the problem in its simplest form.	The problem situation is reduced to the relevant factual situation.
Empirical observation of the problems in the object system	Observation is against a background of facts, which are either known (experience) or given (user supplied), and prior knowledge. The observation must also be focused using a specific set of objectives.
Problem analysis	This involves data collection concerning the problem to understand the true nature of the problem, and to clarify the requirements.
Hypothesis formulation	The analyst now postulates reasons for the problems and determines possible solutions to each problem based on experience, reading, interaction with other analysts, visits to other sites, and so on.
Solution deduction	The analyst now eliminates the incorrect hypotheses and refines the remainder. A conceptual solution to the system is developed based on those hypotheses.
Empirical testing	The proposed system is simulated to establish its feasibility and to prove that it is valid in terms of problem elimination. A prototype of the proposed solution is developed or it is mathematically simulated.
Generalization	The system is mapped onto the live environment as a final test.

Figure 1 Scientific method and systems analysis equivalents

Analysts are often warned not to "prejudge" a situation, but in many ways a prejudgment is an hypothesis. The fault lies not with the prejudgment, but with an inadequate, non-rigorous and subjective testing of that prejudgment prior to development of that system. As Beveridge so aptly states: "Probably the main characteristic of the trained thinker is that he does not jump to conclusions on insufficient evidence as the untrained man is inclined to do." [2]

8. Conclusion

This paper has attempted to define the relationship between scientific enquiry and systems analysis. By deduction it has shown that information systems analysis cannot be placed in the scientific domain primarily because of the object being studied and the non-repeatability of the solution. Despite this, elements of the scientific method can be integrated into the systems analysis procedure to improve the rigour of the process and the correctness of the result. The two sections that most easily lend themselves to improvements are problem verification and solution testability. If systems analysis was a science, it could be argued that no solution should be designed and developed until tested conceptually and proved empirically. However, just because it is not, is no excuse to discard the

principles of scientific method and thereby ignore the aspects of conceptual testing and proof. This implies a detailed understanding of prototyping techniques and a very clear definition of experimental objectives (a useful by-product is that this forces modularity on the solution).

If it is accepted that the purpose of analysis (in the main) is to understand a problem, then the tools and techniques used by the analyst must be problem-directed. An examination of those currently used such as Data Flow Diagrams, Structure Charts, and Activity Charts reveals that they are purely descriptive in terms of process or data flow. They are not analytical and do not drive the focus of the analysis at potential problems. Further research into the analytical process should undoubtedly focus on problem analysis and the elimination of problems in system proposals by validating the conceptual solution.

The close relationship between systems analysis and research indicates that an important component in the education of analysts might be the scientific approach to research and even discussion of the philosophical nature of rigour, repeatability, falsifiability and objectivity. A more detailed understanding of these aspects in relation to the systems analysis domain will improve the process and perhaps systems analysts will one day be able to prove scientifically that for any given problem they have developed "the best solution".

References

[1] N. Ahituv and S. Neumann, [1986], *Principles of Information Systems for Management*, 2nd Edition, Wm C Brown, Dubuque, Iowa.

[2] W.I.B. Beveridge, [1950], *The Art of Scientific Investigation*, WW Norton & Co, New York.

[3] P. Checkland, [1981], *Systems Thinking, Systems Practice*, University Tutorial Press, Suffolk.

[4] J.D. Couger and R.W. Knapp, [1974], *Systems Analysis Techniques*, John Wiley and Sons, New York.

[5] M.J. Culnan and E.B. Swanson, [1986], 'Research in Management Information Systems, 1980-1984 : Points of Work and Reference', *MIS Quarterly* **10** (3), 289-302.

[6] G.B. Davis and M.H. Olsen, [1984], *Management Information Systems, Conceptual Foundations, Structure and Development*, 2nd Edition, McGraw-Hill, New York.

[7] W.S. Davis, [1983], *Systems Analysis and Design - A Structured Approach*, Addison-Wesley, Reading Massachusetts.

[8] V.C. Hare, [1967], *Systems Analysis: a Diagnostic Approach*, Harcourt, Brace & World Inc.

[9] B. Hodge and J.P. Clements, [1986], *Business Systems Analysis*, Prentice-Hall, Englewood Cliffs, NJ 07632.

[10] J. Howard and J.A. Sharp, [1983], *The Management of a Student Research Project*, Gower Publishing Co, Aldershot, Hants GU11 3HR England.

[11] B. Ives and S. Hamilton and G.B. Davis, [1980], 'A Framework for Research in Computer-based Management Information Systems', *Management Science*, **26** (9), 910-934.

[12] R.A. Johnson and F.E. Kast and J.E. Rosenzweig, [1967], *The Theory and Management of Systems*, McGraw-Hill, Tokyo.

[13] A.M. Lee, [1970], *Systems Analysis Frameworks*, MacMillan, London.

[14] P.D. Leedy, [1980], *Practical Research - Planning and Design*, MacMillan Publishing Co, New York, USA.

[15] P.S. Licker, [1987], *Fundamentals of Systems Analysis*, Boyd and Fraser Co, San Francisco.

[16] R.E. Leslie, [1986], *Systems Analysis and Design - Method and Invention*, Prentice-Hall, New Jersey.

[17] H.C. Lucas, [1985], *The Analysis, Design, and Implementation of Information Systems*, McGraw-Hill Book Co, Singapore.

[18] P.B. Medawar, [1969], *Induction and Intuition in Scientific Thought*, American Philosophic Society, Philadelphia.

[19] F.S.C. Northrop, [1947], *Logic of the Sciences and the Humanities*, Greenwood Press, Connecticut.

[20] K. Popper, [1959], *The Logic of Scientific Discovery*, Hutchinson, London.

[21] G.I. Susman and R.D. Evered, [1978], 'An Assessment of the Merits of Action Research', *Administrative Science Quarterly*, **23**, 582-603.

[22] P. Vitalari and G.W. Dickson, [1983], Problem Solving for Effective Systems Analysis: An Experimental Exploration', *Communications of the ACM*, **26** (11), 948-956.

An Approach to Defining Abstractions, Refinements and Enrichments

D G Kourie
Department of Computer Science, University of Pretoria, Pretoria 0083

Abstract

A proposal for defining abstractions and refinements is given in terms of three-valued logic applied to a domain of discourse consisting of a property and an entity set. Definitions for several related concepts flow naturally from these, including possible orderings on refinements and abstractions, as well as the notions of non-determinism, enrichment and base abstractions.
Keyword: *abstraction, refinement, enrichment, three-valued logic, non-determinism.*
Computing Review Categories: *D.2.1, D.2.8, D.3.1, F.3.0, I.2.6*

Received September 1988, Accepted January 1989

1. Introduction

Most software development methods proposed during the last two decades or so depend in one way or another on the idea of stepwise refinement of abstractions. The notion of abstraction also comes to the fore in the Artificial Intelligence context [1,2,3].

Often the concepts of abstraction and refinement are referred to in the literature in an intuitive and/or imprecise way. In other cases definitions, sometimes differing from one author to the next, have been proposed. Such proposals are, however, either limited to a specific context (such as the development of sequential programs), or a definition is given for one of the concepts but not for the other. It would seem, therefore, that there is a need to develop a unified formal theory of abstractions and refinements so as to sharpen and shape the ideas of those who are involved in software development and in the practice of AI. Towards this end, the present paper proposes generic definitions for these concepts, and for several concepts directly associated with these. In particular, the notion of enrichment seems to belong naturally with that of refinement. However, the emphasis is primarily, but not exclusively on software development, rather than on AI.

The next section gives the basic model within which the definitions are made. The definitions of abstractions, refinement and enrichment are given in section 3 as well as a number of corollaries that follow from these definitions. Section 4 is devoted to possible orderings which arise by virtue of the definitions. The final section briefly discusses some of the more important literature that relates to the present work.

2. Basic Model

2.1 Notation and Conventions

The following notation is used :

$X \cup Y$	: the union of sets X and Y
ϕ	: the empty set
iff	: if and only if
$\vdash P$	: Predicate P is true
$P \rightarrow Q$	: Predicate Q logically follows from predicate P

Atomic formulae are predicates or well-formed formulae (wff's) in first order predicate calculus without connectives. A literal is either an atomic formula or an atomic formula preceded by a negation symbol. A clause is the disjunction of one or more literals (cf. Nilsson [9]).

Where it is clear from the context, a set of clauses will denote their conjunction, i.e. $\{P_i \mid P_i$ *a clause*, i = 1..n$\}$ denotes the predicate P_1 & P_2 ...& P_n.

2.2 Domain of Discourse

The context for the discussion below is a *domain of discourse* involving entities and their associated properties. A domain of discourse is represented by a pair of sets : an entity set and a property set respectively. The pair is denoted by <*Ent*,*Prop*>. Apart from being non-empty, no assumption about the cardinality of these sets is made.

Each $E_i \in Ent$ is a constant symbol which is interpreted as a distinct *entity* of interest in the domain of discourse.

Each $p_j(E) \in Prop$ is a clause in which every literal contains the free variable, E, as one of its arguments. E represents some uninstantiated entity in *Ent*. An interpretation is assumed that identifies each literal of each clause $p_j(E)$ (and therefore each clause as a whole) as a *property* of the uninstantiated entity E. For example, the

appearance of the free variable E in a literal such as *colour(E,blue)* is construed to mean that the uninstantiated entity E has the colour blue.

When E is substituted by entity $E_i \in Ent$, the resulting clause is denoted by $p_j(E_i)$. $p_j(E_i)$ is called a *property assertion* of entity E_i. It is a clause that asserts that the specific entity E_i possesses the property alluded to in $p_j(E)$.

In general, only properties which are considered directly relevant in the domain of discourse are included in *Prop*. However, it will be assumed that if a clause is considered relevant then each separate literal in the clause together with its negation is also in *Prop*.

An interpretation of property assertions in the domain of discourse is assumed to exist in the form of a total function :

$I : Ent\ X\ Prop \rightarrow \{true, false, unknown\}$.

The notation $\vdash p_j(E_i)$, or $\vdash \sim p_j(E_i)$, or $?p_j(E_i)$ will be used for $I(E_i,p_j(E))$, depending on whether $I(E_i,p_j(E))$ has value *true* or *false* or *unknown* respectively. It is assumed that the interpretation I is consistent with the normal (strong) truth tables that apply in 3-valued logic [6]. Hence, for example, the interpretation of a property assertion cannot be that it has truth value of false or unknown if the interpretation of one of its disjuncts is that it has a truth value of true.

Each entity, E_i, thus partitions *Prop* into three sets called the true, unknown or false property sets of E_i, which are respectively denoted by:

$true(E_i)$
$= \{p_j(E) \,/\, p_j(E) \in Prop \ \&\ \vdash p_j(E_i)\}$

$unknown(E_i)$
$= \{p_j(E) \,/\, p_j(E) \in Prop \ \&\ ?p_j(E_i)\}$

$false(E_i)$
$= \{p_j(E) \,/\, p_j(E) \in Prop \ \&\ \vdash \sim p_j(E_i)\}$

It can easily be shown, using the assumption of consistency in 3-valued logic, that if *Prop* and $true(E_i)$ are given, then it is possible to infer $false(E_i)$ and $unknown(E_i)$. Hence, where no further qualification is made, reference to a property set should be taken to indicate the true property set.

The foregoing assumes that the truth value of a property assertion in a domain of discourse is fixed over time, space, environment, etc. It is either true (or false) under all circumstances (or modalities), or else it has value 'unknown' – even it is known to be true (or false) in some circumstances.

Henceforth it will also be assumed that every element of *Prop* is a member of the true property set of at least one entity in *Ent*. Furthermore, we shall say that entity E_i has or does not have the property $p_j(E)$ according as to whether $p_j(E) \in true(E_i)$ or not.

It is interesting to note that entities may be characterized as either deterministic or non-deterministic in terms of the foregoing model. An arbitrary entity $E_i \in Ent$ is *deterministic iff* $unknown(E_i) = \phi$. An entity which is not deterministic is *non-deterministic*.

3. Definitions

3.1 Property Definitions

In this section two arbitrary properties $p_x(E)$ and $p_y(E)$ in *Prop* are considered, and for conciseness they are denoted by x and y respectively.

The notation $x \rightarrow y$ is used to assert that y logically follows from x. A necessary condition for this to hold is that every entity in *Ent* which has property x (and by assumption there will be at least one such entity) also has property y.

Conversely, $\sim[y \rightarrow x]$ asserts that y does not logically follow from x. A sufficient condition for this to hold is that some entity in *Ent* which has property y (and again by assumption there will be at least one such entity) does not have property x.

The following definitions will be useful :

- Property x *refines* property y *iff* $\vdash ([x \rightarrow y] \ \&\ \sim[y \rightarrow x])$
- Properties x and y are *equivalent iff* $\vdash ([x \rightarrow y] \ \&\ [y \rightarrow x])$
- Properties x and y are *independent iff* $\vdash (\sim[x \rightarrow y] \ \&\ \sim[y \rightarrow x])$
- Properties x and y *enrich* one another *iff* they are independent, but are common properties of at least one entity in *Ent*.
- Properties x and y are *mutually exclusive iff* they are independent but do not enrich one another.

Hence, any two properties in the property set of a given entity are either equivalent, or they are independent and enrich one another, or one of them refines the other.

However, equivalent properties will always be either jointly present or jointly absent in an entity's property set. They are thus not helpful in discriminating between entities. Consequently, it will be assumed that domains of discourse are chosen in such a way that no equivalent properties occur. Thus any pair of properties that an entity has either enrich one another, or one of them refines the other.

3.2 Abstraction

Consider two distinct entities E_i and E_j in *Ent*. E_j is defined as an *abstraction* of E_i *iff* $true(E_j)$ is a subset of $true(E_i)$.

This definition implies the following about E_i and E_j:

- Since E_i and E_j are distinct entities, the following must hold :
  - $true(E_j)$ is a proper subset of $true(E_i)$,
  - $unknown(E_i)$ is a proper subset of $unknown(E_j)$,
  - $false(E_j)$ is a proper subset of $false(E_i)$.
- Properties in $true(E_i)$ but not in $true(E_j)$ are in $unknown(E_j)$. Similarly properties in $false(E_i)$ but not in $false(E_j)$ are also in $unknown(E_j)$. Informally, therefore, there is a parallel between abstraction and non-determinism. The degree of abstraction is directly related to the degree of non-determinism.
- An entity may be an abstraction of several entities which may be, but need not be abstractions, of one another.

3.3 Refinement/Enrichment

Based on these notions, an entity may sometimes be regarded as a refinement or an enrichment of one or more of its abstractions. Consider the entity E_r and one of its abstractions, E_a. Let $D = true(E_r) - true(E_a)$, i.e. D is the set of properties that E_r has in addition to those in E_a.

- Entity E_r is a *refinement* of entity E_a *iff* every property in D refines some property in $true(E_a)$.
- Entity E_r is an *enrichment* of entity E_a if every property in D enriches *all* properties in $true(E_a)$.

Note that if some (but not all) properties in D refine properties in $true(E_a)$, and the remaining properties in D enrich all properties $true(E_a)$ then E_r neither refines nor enriches E_a.

3.4 Corollaries

Several corollaries follow from the foregoing definitions. A non-exhaustive list includes the following :

- The entity set may be partitioned into two sets: a set of entities which are abstractions of other entities and a set containing the remaining entities. Elements of the latter set may appropriately be called *concrete* entities in the domain of discourse, and elements of the former, *abstract* entities.
- The enrichment relationship for properties is reflexive, but not transitive. For entities the enrichment relationship is irreflexive and transitive.
- The refinement relationship is transitive but irreflexive for both entities and properties.
- Furthermore, if $p_k(E)$ refines $p_l(E)$ and $p_k(E) \in true(E_j)$, then $p_l(E) \in true(E_j)$. However, the converse does not necessarily hold, namely if $p_k(E)$ refines $p_l(E)$ and $p_l(E) \in true(E_j)$, then it does not follow that $p_k(E) \in true(E_j)$.
- Entity refinement and entity enrichment are mutually exclusive notions.
- If an entity E_a is an abstraction of entity E_r, and latter has exactly one more property in its property set than the former, then E_r either refines E_a or E_r enriches E_a.
- Let E_a be some arbitrary abstraction of E_r and let E_{ra} denote some abstraction of E_r which has E_a as one of its abstractions. By choosing E_{ra} appropriately, at least one of the following assertions will be true :
  a) E_r refines E_a
  b) E_r enriches E_a
  c) E_{ra} refines E_a
  d) E_{ra} enriches E_a
- If a) holds above, then so does c) for *any* choice of E_{ra} and similarly for b) and d).

3.5 Base Abstractions

If a given property in *Prop* refines another in *Prop*, it will be called a *refining* property. All those properties in *Prop* which are not refining are called *base* properties.

Note the following :

- Every pair of base properties in the property set of an arbitrary entity enrich one another.
- A refining property always refines at least one base property, but may also refine one or more other refining properties.
- The property set of any entity E_i can be uniquely partitioned into a non-empty set of base properties, and a (possibly empty) set of refining properties. These sets will be denoted by $bas(E_i)$ and $ref(E_i)$ respectively.

An abstraction E_j of E_i which is such that $true(E_j) = bas(E_i)$ will be called a *base abstraction* of E_i, and will be denoted by $ba(E_i)$.

However, while every entity has at most one base abstraction, a base abstraction need not always exist. For example, if $ref(E_i) = \phi$, then $true(E_i) = bas(E_i)$, and no base abstraction of E_i exists. Also, $bas(E_i)$ may be such that while individual properties in this set enrich one another, the consistency requirement for 3-valued logic truth tables demands that other refining properties always be present in the true property set of any abstraction of E_i. For example, properties of the form $(a \text{ or } b)$ and $(\sim b)$ enrich one another (neither logically follows from the other), but any abstraction with these properties in its property set, necessarily also contains the refining property (a) in its property set.

Nevertheless, in applying progressive refinement and/or enrichment to arrive at the properties of an entity E_i, much of the initial creative effort will go into determining $bas(E_i)$, and indeed into the base abstraction of E_i if it exists.

4. Ordering

Since refinement of entities is transitive and irreflexive, it defines a strict partial order on entities of *Ent*, which can be denoted by $>_r$. Hence if entity E_p is a refinement of entity E_q, this may be written as $E_p >_r E_q$.

Similarly enrichment of entities is a transitive and irreflexive relation, and and defines a strict but partial order $>_e$ on entities of *Ent*. Thus $E_p >_e E_q$ denotes the fact that E_p is an enrichment of E_q.

The number of base properties in the property sets of entities provides a basis for defining a weak partial order on abstractions of entities in *Ent*. The order will be denoted by $\geq_b$. Given two arbitrary abstractions, E_p and E_q, of an arbitrary entity E_c (and allowing also for the possibility that either E_p and/or E_q is the entity E_c itself) then $E_p \geq_b E_q$ *iff* the number of base properties in $true(E_p)$ is greater or equal to the number of base properties in $true(E_q)$. $E_p \geq_b E_q$ is read as: 'E_p is at least as basic as E_q'.

Note the following :

- $E_p >_e E_q \rightarrow (E_p \geq_b E_q)$ & $\sim(E_q \geq_b E_p)$ since an enrichment, per definition, has more base properties than the entity which it enriches.
- $E_p >_r E_q \rightarrow (E_p \geq_b E_q)$ & $(E_q \geq_b E_p)$ since a refinement of an entity per definition adds refining properties, but not base properties, to the property set of the refined entity.

In a given context some base properties may be regarded as more important that others. In such a case, a suitably chosen weighting could be applied to the base properties and the ordering would then be according to the weighted sum of base properties.

Several approaches to defining orderings on abstractions now suggest themselves, some of which are more naive than others.

The most naive approach is to postulate a weak total order on abstractions, based on the number of properties in the respective true property sets. The semantics of such an ordering is, however, not very clear.

A somewhat more meaningful approach is to postulate a strict partial order, $<_a$, such that $E_p <_a E_q$ *iff* E_q is an abstraction of E_p.

Ideally an ordering for any pair of abstractions of the same entity would be desirable – even if neither one is an abstraction of the other. One approach for achieving this would be to extend the previous ordering to a weak order, $\leq_a$, whereby:

$E_p \leq_a E_q$ *iff*

$(E_p <_a E_q)$ or (I)

$(\sim E_p <_a E_q$ & $\sim E_q <_a E_p$ & $E_p \geq_b E_q)$. (II)

In general, such an ordering is weak when considered over all entities in *Ent* since neither (I) nor (II) need necessarily hold for two arbitrary entities. However, the ordering is total over abstractions of a given entity.

Note that if (II) holds, but not (I), then the ordering implies that an abstraction with few base properties which may possibly have many refining properties is to be considered more abstract than an abstraction with many base properties which may not be at all refined. Furthermore, two abstractions with disjoint property sets each having the same number of base abstractions would be considered to be equally abstract, even if the property set of one abstraction contained many refining properties, while the other contained none. In other words the number of base properties, rather than the degree of refinement dominates in determining the level of abstraction.

An even more sophisticated ordering of abstractions could be defined based on a metric on each abstraction. This metric should reflect not only the number of base properties (perhaps weighted in some way) in the relevant property set, but also the extent to which the property set shows how individual base properties have been refined. Precisely how such a metric should balance the degree of refinement against the number of base properties will depend, inter alia, on the domain of discourse.

5. Other Work

This paper is a summary of work reported on elsewhere [7]. The latter extends the concepts discussed here to the area of specifications, and proposes definitions for concepts such as valid or invalid, partial or complete specifications, as well as for abstract, refined or enriched specifications.

The notion of abstraction is qualitatively explained in a number of texts on software development. A typical example is :

> *'By abstraction we mean the act of singling out a few properties of an object for further studies or use, omitting from consideration other properties that don't concern us for the moment.'* [4]

Darden [2] spells out a similar idea of abstraction in the AI context in somewhat more detail, claiming that 'abstraction formation involves loss of content'. The present paper's definition of an abstraction is very much in the spirit of these definitions. It is, however, more formal, and given in the context of a domain of discourse.

Benzon and Hayes [1] criticize Darden's approach. However, the latter appear to be more concerned with 'issues related to human cognitive

endeavors in forming abstract concepts' rather than doing 'a conceptual analysis of abstraction as it is used in current computational AI work' [3].

Hoare *et. al.* [5] propose an weak partial ordering between programs which is very similar to the ordering on abstractions $<_a$ discussed above. Their ordering, which will be denoted here by $\geq_H$ is defined as :

$P \geq_H Q$ *iff* $P \cup Q = P$

where $P \cup Q$ designates a program that makes a non-deterministic choice to function as either P or as Q. Were it not for the fact that in the present paper $<_a$ has been defined irreflexively, the following would hold for any pair of programs P and Q:

$P \geq_H Q$ *iff* $Q <_a P$.

The definition of $\geq_H$ leads to the concept of an *abstract program*, a concept that is stricter but similar to the notion of an abstract specification [7].

Morgan and Robinson [8] propose a definition for refinement in the context of software development which is based on the concepts of pre- and post-conditions. Using the well-known notation *wp(P,post)* for the weakest precondition of a program P with post-condition *post*, they define refinement as follows:

> *'For programs P and Q we say P is refined by Q, written* $P \leq Q$, *iff for all post-conditions post : wp(P,post) => wp(Q,post).'*

It can be shown that if Q is a refinement of P (but not vice-versa) according to the definition of Morgan and Robinson then Q is a refinement of P for a suitably chosen domain of discourse, according to the present paper's definition [7].

To the author's knowledge the theme of enrichment has not been formally addressed in the literature on software development.

References

[1] W Benzon and D Hayes, [1987], *Reactions to Darden*, Letter in AI Magazine, **8**(4), 7-8.

[2] L Darden, [1987], *Viewing the History of Science as Compiled Hindsight*, AI Magazine, **8**(2), 33-41.

[3] L Darden, [1987], *Darden's Response*, Letter in AI Magazine, **8**(4), 11.

[4] D Gries, [1981], *The Science of Programming*, Springer-Verlag.

[5] C A R Hoare, I J Hayes, J He, C C Morgan, A W Roscoe, J W Sanders, I H Sorensen, J M Spivey and B A Sufrin, [1987], *Laws of programming*, CACM, **30**(8), 672-686.

[6] S C Kleene, [1974], *Introduction to Metamathematics*, Wolters-Noordhof Publishing and North Holland Publishing Company, 7th Reprint.

[7] D G Kourie, [1988], *Towards a general theory of abstractions, refinements and enrichments*, Internal Report, University of Pretoria.

[8] C Morgan and K Robinson, [1987], *Specification statements and refinement*, IBM J. Res. Develop., **31**(5), 546-555.

[9] N J Nilsson, [1982], *Principles of Artificial Intelligence*, Springer-Verlag.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. These may include research, review and exploratory articles of interest to the journal's readers. The preferred language of the journal will be English, although papers in Afrikaans or other congress languages of IFIP will not be precluded. Typed manuscripts for review should be submitted in triplicate to:

Professor D G Kourie
Department of Computer Science
University of Pretoria
Hatfield 0083
Pretoria

Form of Manuscript

Manuscripts for review should be prepared according to the following guidelines.

- Use double-space typing on one side only of A4 paper, and provide wide margins.
- The first page should include:
  - title (as brief as possible)
  - author's initials and surname
  - author's affiliation and address
  - an abstract of less than 200 words
  - an appropriate keyword list
  - a list of Computing Review Categories.
- Tables should be typed on separate sheets of A4 paper, and should be numbered and titled.
- Figures should also be supplied on separate sheets of A4 paper, and should be identified on the back in pencil with the author's name and the figure number. Original line drawings, and not photocopies, should be submitted.
- Mathematical and other symbols may be either handwritten or typed. Greek letters and unusual symbols should be identified in the margin. Distinguish clearly between:
  - upper and lower case letters
  - the letter O and figure zero
  - the letter I and the number one
  - the letter K and kappa.

  References should be listed after the text in alphabetical order of the (first) author's surname, cited in the text in square brackets. References should take the following form:
  [1] E Ashcroft and Z Manna, [1972], The translation of 'GOTO' programs to 'WHILE' programs, *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255.
  [2] C Bohm and G Jacopini, [1966], Flow diagrams, Turing machines and languages with only two formation rules, *Comm. ACM*, 9 (3), 366-371.

Manuscripts accepted for publication should comply with the above guidelines, but may be in one of the following three formats:

- typewritten and suitable for scanning
- provided as an ASCII file on diskette
- camera-ready.

Authors wishing to provide camera-ready copy may obtain a page specification from the production editor.

Charges

A page charge, scaled to reflect production costs, will be levied on papers accepted for publication. The costs per final page are as follows:

Typed format, not camera-ready R60
Disk in ASCII format R40
Camera-ready format R20

These charges may be waived upon request of the author, and at the editor's discretion.

Proofs

Proofs of accepted papers will be sent to the author to ensure that typesetting is correct, and not for addition of new material or major amendments to the text. Corrected proofs should be returned to the production editor within three days.

Copyright

Copyright in published papers will be vested in the publisher.

Letters and Communications

Letters to the editor will be welcomed and will provide a forum for discussion on topical issues. They should be signed, and should be limited to about 500 words.

Communications reflecting minor research contributions will be considered for publication in a separate section of the journal. Such communications will, however, not be regarded as a fully-fledged publication for FRD subsidy purposes.

Book Reviews

Contributions in this regard will be welcomed. Views and opinions expressed in such reviews should, however, be regarded as those of the reviewer alone.

Advertising

Placement of advertisements at R1000 per full page per issue and R500 per half page per issue will be considered. Enquiries should be directed to the production editor.