ICL TECHNICAL JOURNAL

Volume 5 Issue 1 May 1986

Published by
INTERNATIONAL COMPUTERS LIMITED
at
OXFORD UNIVERSITY PRESS

ICL TECHNICAL JOURNAL

The ICL Technical Journal is published twice a year by International Computers Limited at Oxford University Press.

1986 subscription rates: annual subscription £16 UK, £19 rest of world, Us $40 N. America; single issues £10 UK, £11 rest of world, US $25 N. America. Orders with remittances should be sent to the Journals Subscriptions Department, Oxford University Press, Walton Street, Oxford OX2 6DP, UK.

Printed by H Charlesworth & Co Ltd, Huddersfield ISSN 0142-1557

ICL TECHNICAL JOURNAL

Contents

Volume 5 Issue 1

Editorial

As will be evident, with this issue we have a change of publisher, the Journal now being published for ICL by Oxford University Press.

I can say without hesitation that we have been very happy with our previous publishers Peter Peregrinus Limited, and I personally have found it a pleasure to work with a series of their editors – David Mackin, Pauline Maliphant and most recently John Cooper – and their supporting staffs. However, we felt that after seven years we should make a change and we now welcome our new association with OUP, with whom I look forward to equally pleasurable collaboration.

In a previous editorial note I said that the Editorial Board was planning to compile more issues concentrating on single themes or topics. The two last issues, for May and November 1985, were both single-theme issues, dealing exclusively, although from many different points of view, with two ICL products, the Series 39 Level 30 mainframe computer and the Content Addressable File Store (CAFS) respectively. Both these proved very popular. The present issue covers a range of subjects, but five of the papers deal with different topics in the broad field of software techniques and technologies. The two by Duce & Fielding and Henderson & Minkowitz are concerned with the use of formal, meaning essentially mathematical, methods for specifying and designing software systems; those by Lowden & de Roek and West with the use of natural language in extracting information from a database; and by Kitchenham et al. with one of the techniques used to detect errors at as early as possible a stage in the development of a large software system.

There is a very active interest in computer history now; Campbell-Kelly's paper is the first of a series in which he will survey the history of research and development in the companies that formed the predecessors of ICL. Dr. Campbell-Kelly has been given access to ICL's archives and is planning a book on the history of the company.

ICL company research and development Part 1 1904–1959

M. Campbell-Kelly
Department of Computer Science, University of Warwick

Abstract

The ICL company of today has its origins in the punched-card machine industry which began early in this century and matured between the two world wars; it was not until well into the 1960s that punched-card accounting machines fully gave way to computers. The paper traces the origins and growth of R & D activity for punched-card machines and early computers, from the turn of the century up to the late 1950s.

1 Introduction

In July 1984 the author entered into an agreement with ICL to write a company history. ICL has roots which go back many years to the punched-card machines from which present-day commercial computers are descended. There has in fact never been a satisfactory history written of the punched-card machine industry, perhaps for the good reason that it was of little economic importance. For example, IBM, outstandingly the most successful punched-card machine company, was only one-thirtieth of its present size (in terms of employees) in the late 1930s; it was successful and profitable, but not of global significance. However, to understand the structure of the present-day computer industry (and to some extent its products) it is necessary to understand the development of the punched-card machine industry. The first phase of the company history project is thus to trace the development of the two British punched-card machine manufacturers – The British Tabulating Machine Co. Ltd. (BTM) and Powers-Samas Accounting Machines Ltd. – from the inception of BTM in 1904 up to the merger of the two companies in 1959 to form International Computers & Tabulators Ltd. (ICT).

The period 1904–1959 saw R & D transformed from what in the early 1900s had been an activity consuming negligible resources to an activity that was a full divisional entity in its own right in a company 17 000 employees strong by 1959. The purpose of this paper is to survey this R & D activity, to discuss the historical sources on which it is based, and to set it in its industrial and economic context.

2 Foundations of the British punched-card machine industry (1904–1919)

The punched-card machine industry had its origins in the machines invented by Herman Hollerith (1860–1929) to process the 1890 census of the USA. Hollerith's equipment included a simple card punch and the 'census machine' – a combined tabulating machine and sorting box (Fig. 1). Although the technology was unsophisticated by later standards, Hollerith's early machines were very effective: some 700 punching machines and 100 census machines were employed by the Bureau of the Census, an impressively large data-processing operation even by today's standards.

In 1896 Hollerith incorporated the Tabulating Machine Company (TMC) in New York for the commercial exploitation of his machines. During the early 1900s he designed an improved range of machines which were soon taken up by railroad concerns, insurance companies, and manufacturers[1]. By about 1907 the machines comprised a key punch (of a basic design still occasionally seen in use), a tabulator and a card sorter. The key piece of machinery was the tabulator, which accumulated values from a pack of cards, the results being transcribed by hand from the counters (Fig. 2). Hollerith was an outstanding inventor-entrepreneur, but it it is only quite recently that his role in the business machine industry has been fully appreciated and a major biography appeared[2].

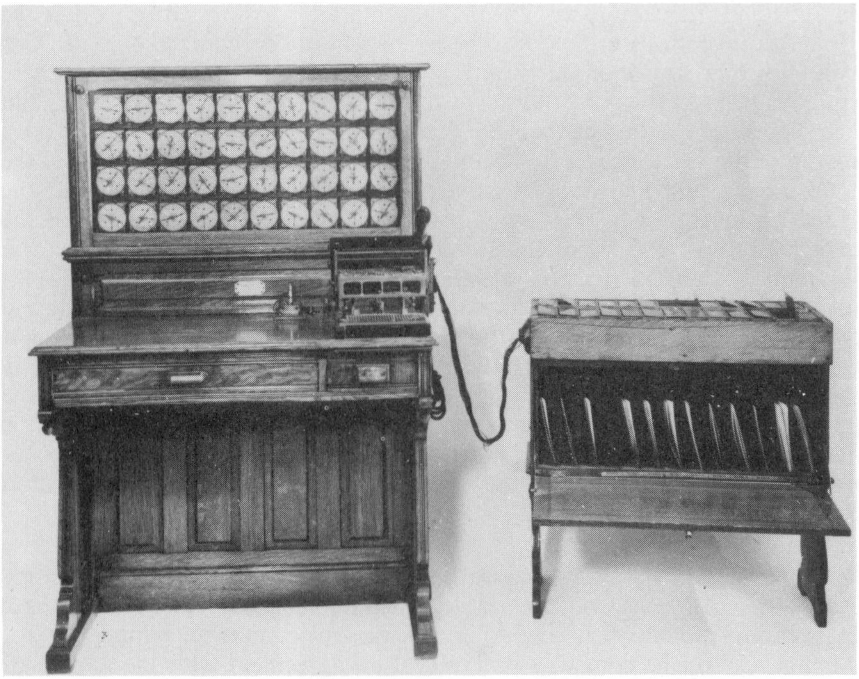


Fig. 1 Census machine, *c.* 1890

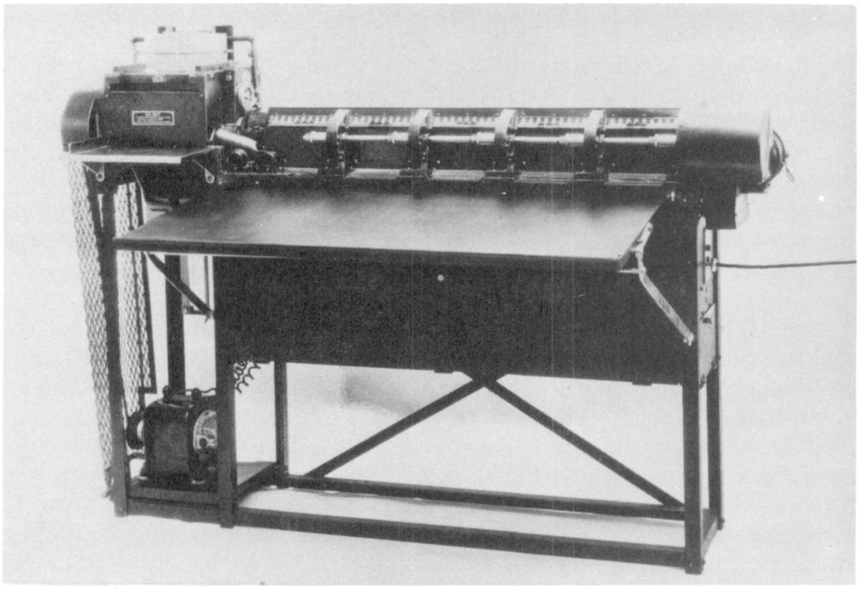


Fig. 2 Hollerith electrical tabulating machine, *c.* 1905

In 1904 a British company 'The Tabulator Ltd.' was formed to exploit the Hollerith inventions in Britain. The formation of this company, some 3000 miles from the source of the invention in the USA, affords an interesting case study in what we would now call 'technology transfer'. At the legal and licensing level the transfer was effected by Robert P. Porter, a British-born citizen of the USA. Porter had been director of the census at the time of the 1890 census, and had been instrumental in bringing about the use of the Hollerith machines by the Bureau. In 1902 he had become editor of the engineering supplement of the London *Times*, and prior to his arrival in England he had contracted with Hollerith for the British rights to his inventions. Porter became a founding director of The Tabulator Ltd. and assigned his rights to the Hollerith patents to the company. The new company was largely guided by Ralegh B. Phillpotts, at that time company secretary of British Westinghouse. At the technical level, the knowhow of the punched card machines was brought to England by C.A. Everard Greene, the first general manager of the British company, who spent some 18 months during 1902–1904 learning to use and maintain the Hollerith machines, and to assemble them from parts.

In 1907 the company was reformed and incorporated as The British Tabulating Machine Co. Ltd. (BTM) with a greatly increased authorised share capital of £50 000, needed to finance the leasing of the machines. One of the first acts of the company was to seal an agreement with the American company in 1908 to exclusively manufacture and sell the machines in Great Britain and the Empire (excluding Canada). Phillpotts (later Sir Ralegh

Phillpotts) was chairman and sometime managing director of BTM from its inception in 1907 until his retirement in his late seventies in 1950. BTM grew at a rather gentle rate from 5 employees in 1907 to 45 employees at the outbreak of World War I in 1914. We have a surprisingly clear picture of these early years, for although very little in the way of ephemeral documentation has survived we do have the statutory annual reports of the company and the minutes of board meetings. In those days, when the company made in the region of 10–20 new installations a year, each new customer was cause for some celebration and was duly recorded in the minute book. Another valuable source is a set of reminiscences written by C.A. Everard Greene in 1959[3] long after his retirement as general manager, which vividly evoked his early years; until ICL opened up its archives for the present company history project, this was the principal documentary source for early company history. Another fascinating source is a delightfully frank bundle of correspondence between Hollerith and the British company written during the early years of the company.

In the USA relations between the Tabulating Machine Company and the Bureau of the Census became strained after the 1900 census, as a result of which in 1907 the Bureau engaged an inventor, James Powers, to design a new range of punched-card machines which enabled it to become independent of the Hollerith company for the 1910 census[4,5]. In 1911 Powers incorporated a company, Powers Accounting Machines, in New York to develop a range of punched-card machines for commerce which were marketed from about 1914. The Powers machines, although functionally similar to Hollerith's, operated on entirely mechanical principles and used a mechanical pin-sensing method that, unlike the Hollerith electrical sensing, was unaffected by the presence of metal impurities in the cards. The Powers printing tabulator (Fig. 3) was a marked improvement on the Hollerith model; by eliminating the manual transcription results it was much more suitable for accounting work.

Powers was an inventor-entrepreneur much in the mould of Hollerith, but he remains an enigmatic figure of whom we know very little. A Russian migrant to the USA, the extant biographical data on him amounts to little more than two typed pages[6]. After the formation of the Powers company the history of the punched-card machine industry was dominated by an intense rivalry between the Powers and Hollerith lines.

In 1915 an American-owned subsidiary of the Powers company was formed, The Accounting and Tabulating Corporation of Great Britain Ltd. – often known as the 'Acc & Tab'. The largest of the British company's early users was the Prudential Assurance Co. No doubt for the twin motives of ensuring independence from an American supplier and also as a profitable and imaginative investment, the Prudential acquired the company in 1919, with rights to manufacture and sell the Powers machines in Great Britain and certain other territories. The commercial rivalry between the Hollerith and Powers machines was thus continued in Britain, and in other countries in which the two companies operated.

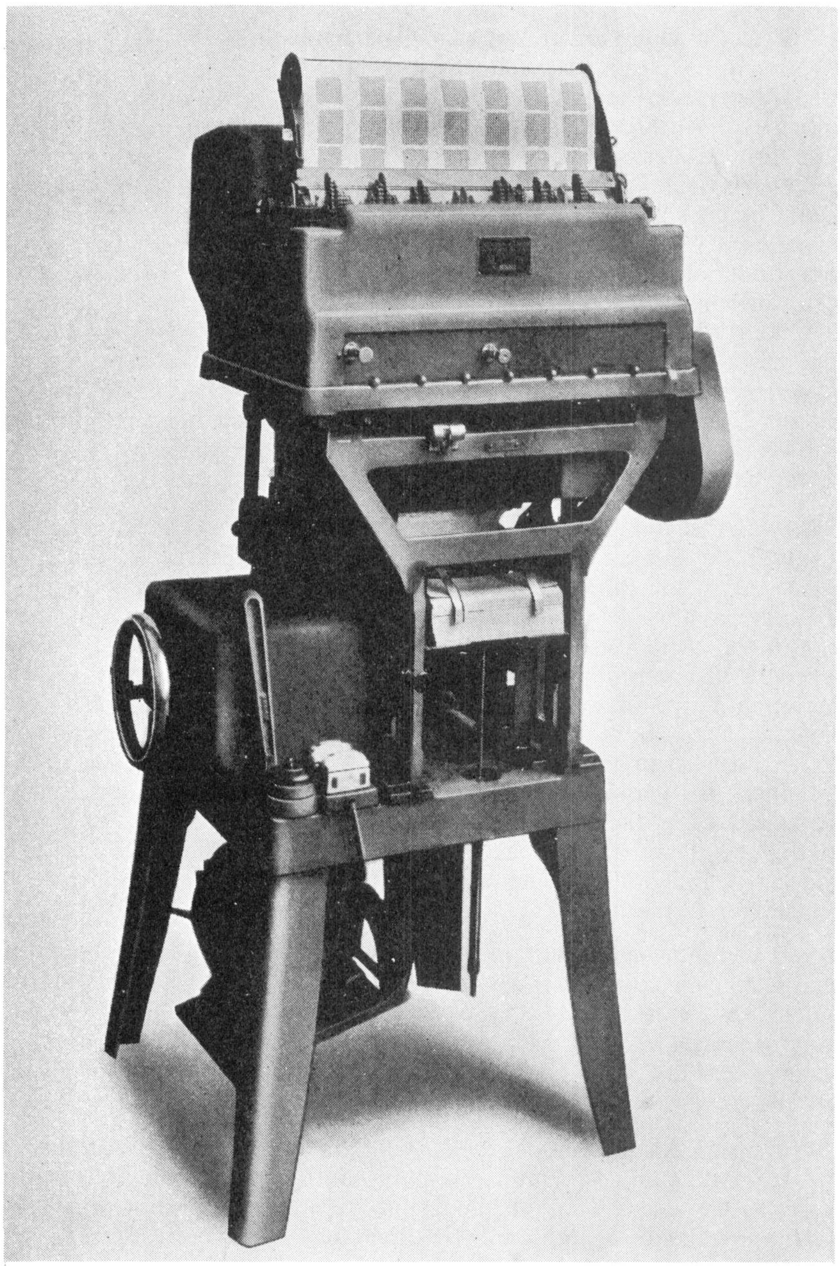


Fig. 3 Powers printing tabulator, *c.* 1914

3 The heyday of the punched-card machine industry (1920–1939)

The period between the two world wars was the heyday of the punched-card machine industry: it was a period in which the technology matured and its products became part of the fabric of the commercial world. The 1920s saw both British companies continue the gentle growth of their early years, barely checked by difficult economic conditions. The more buoyant trading conditions of the 1930s, combined with a growing acceptance of office mechanisation, saw dramatic growth in both companies. Table 1 gives some indication of the growth of BTM between the wars by charting the growth of capital employed. (This data is taken from the statutory annual reports; equally good measures would be the number of employees or factory output, but complete records do not appear to have survived.)

As the companies expanded in size, so they became more hierarchical and structured. In the earliest days any employee, from the general manager down, would have been capable not only of the assembly and maintenance of machines but also of technical development work and the analysis of customer requirements. But by the 1920s both companies had fragmented into sales and production divisions with supporting accounting and secretarial functions.

Table 1 BTM capital employed

Year	Capital
1910	£12 000
1915	£25 000
1920	£70 000
1925	£123 000
1930	£159 000
1935	£302 000
1940	£681 000

Up to the 1920s both British punched-card machine companies had been in essence marketing and maintenance operations for machines of largely American manufacture. For both companies the transition to British manufacture, and eventually British design, was a gradual one. In the case of BTM the direct importation of complete machines had already given way to the assembly of machines from American parts, with some local content, before the outbreak of World War I. The depression years of the 1920s, however, saw the emergence of a vocal 'Buy British' movement that expedited BTM's move towards increased local manufacture: a new factory was opened in Letchworth in 1921 which expanded severalfold in the next few years. The company still continued to import American machines for which the volume of demand could not justify tooling-up for local manufacture.

The Acc & Tab made the transition from importing to manufacturing machines a good deal more quickly than did BTM. This was no doubt forced upon it to some degree by the American parent's rather indifferent record of product innovation and quality of manufacture. In 1919 the company opened a factory in Aurelia Road, Croydon, and rapidly moved towards wholly British designed and manufactured machines*. In the USA the parent

*There is a remarkable silent film of the factory made in 1926 which has been restored by the National Film Archive in recent years.

company was under threat of liquidation in 1923 and this no doubt encouraged further expansion and independence.

It is a tribute to the vigour of the British Powers company that it was able, in the absence of a strong American parent, to hold its own against BTM; the two companies remained of comparative size and profitability through to the 1950s. In the USA the picture was very different. In 1914 Thomas J. Watson became president of TMC and under his energetic leadership it soon eclipsed the Powers organisation both in terms of product innovation and in sales[7]. In 1924 the company's growing confidence and success in foreign markets was reflected by a change of name to International Business Machines; at the same time the fortunes of the Powers company were at a very low ebb indeed. In 1927 the foundering Powers company was acquired in the merger which resulted in the formation of the Remington-Rand company. As a result of the merger the company's products and profitability were revived; but although a serious competitor for IBM, it was never to achieve the success of its rival†.

4 Technical development 1920–1939

When Watson assumed control of TMC in 1914, one of his first priorities was to develop a printing tabulator to compete with the much superior machine offered by the Powers company. To this end he established a development laboratory at Endicott, New York, whose staff was eventually to include such outstanding inventors as J.W. Bryce, F.M. Carroll and C.D. Lake. The IBM research department, and those developed by the British companies later, were of a pattern that had ceased to exist by the 1950s when research and development had become completely professionalised. The staff of these early research departments – inventors rather than scientists – were typically men of little formal education who had risen through the ranks. They included people of outstanding talent whose contribution even now is little appreciated: J.W. Bryce, for example, who was one of IBM's most prolific inventors, died in 1949 with scores of patents to his name[9].

The printing tabulator (designed by Clair Lake) duly emerged from Watson's research department, and IBM company lore has it that when the machine was demonstrated to the 1919 sales conference the salesmen stood on their chairs and cheered[10]. The new tabulator was marketed in the USA from 1921, but it was not until 1924 that IBM supplied machines for the British market.

R & D activity did not become formally established in the British companies until the early 1920s, although there was some *ad hoc* development and patent activity prior to this time. A notable example within BTM occurred in connection with the tabulators modified for the 1911 British census, of which

†Reliable data is hard to come by, but a contemporary report[8] suggests that at that time IBM had five installations in the field for every one of Remington-Rand's.

a detailed account appeared in the journal *The Engineer*[11] ‡.

Probably the most significant invention in the development of punched-card machinery, and unquestionably the most important British contribution, was made by an Acc & Tab engineer Charles C. Foster: the alphabetical printing unit[12,13]. Prior to the introduction of alphabetical machines, tabulators had only been able to print figures; now it became possible to print names and addresses, product descriptions and so on, completely transforming the range of possibilities for punched-card accounting. Although the original patent was applied for in 1916 (British patent 108942) development was hindered by World War I and a prototype tabulator was not publicly demonstrated until 1921. In a dramatic and rare reversal of the flow of technology transfer, the American Powers company eagerly exploited the Foster patent in the United States. Foster became one of the best known figures in the British Powers company, and at the time of his retirement in 1955 his contributions received official recognition by the award of an MBE.

In 1922 BTM formally established its 'Experimental Department' at Letchworth under Charles Campbell, one of the company's first and most able technical employees. Machine development was only one aspect of the work of the department; it was also responsible for commissioning non-standard tabulators and making preproduction prototypes, as well as training maintenance staff. A similar pattern developed in the British Powers company, where the first head of the engineering department was Arthur Thomas, another powerful figure in British punched-card machine development.

We actually know very little at a detailed level about punched-card machine development during the 1920s for either company, because very few records have survived; nor do there appear to be any surviving people then at a senior level to ask. Moving to the 1930s the picture becomes just a little clearer, at least for the Powers company for which there exists a single volume of surviving documentation covering the years 1930–1933 of the 'Development Committee'. This committee was instigated in 1930 to meet approximately monthly to co-ordinate technical development from the viewpoints of both the production and selling organisations. Between the years 1930 and 1944 the Development Committee met 161 times: the minutes and supporting documents for meetings 1–39 and 113–161 are all that survive, giving a tantalisingly incomplete picture of development. This, however, is a mine of information compared with the negligible documentation that has survived on BTM development activity.

Nonetheless, the broad sweep of technical development is immediately apparent from the succession of products marketed by the Hollerith and Powers companies in America, in Britain and elsewhere. The picture is one of

‡Sir Gerald Chadwyck-Healey, who was a director of BTM, was also chairman of *The Engineer*. The journal carried several articles on Hollerith machines over the years, but never one on Powers machines!

intense rivalry between the companies, in which each company tried alternately to better the products of its competitor. A well known example was the introduction in America in 1928 of the 80-column punched card with slotted holes in place of the standard 45-column card with round holes then offered by both companies. The greater capacity of the new card was a challenge that had to be swiftly met by Remington-Rand, which offered a 90-column card shortly after. The British companies adopted the new size cards within a year or two of their appearance on the American market.

One of the notable British achievements during this period was the small card introduced in 1932 by Powers-Samas§. The new size card ($4\frac{5}{8} \times 2$ in instead of the $7\frac{3}{8} \times 3\frac{1}{4}$ in of the full-sized card) enabled a range of low-cost machines to be put on the market for small and medium-sized businesses, which could not have justified the large machines. These machines soon became a mainstay of the company and sold in their hundreds. BTM quickly countered with their own 'Junior' range of machines, but they were not a great success. Another important British first was the 'rolling total' mechanism developed under H.H. ('Doc') Keen, one of BTM's outstanding technical people. The rolling total feature enabled values to be 'rolled' from one counter to another, considerably enhancing the arithmetical capabilities of the machine. The alphabetical rolling total tabulator launched in 1936 (Fig. 4) was a high point in prewar British punched-card machine design, and functionally superior to the contemporary IBM machine.

Over the years a host of improvements were made to the machines. These improvements included major advances such as rolling totals, the 'automatic total attachment' and the 'long feed mechanism', as well as a host of minor improvements to counter, printing, and card sensing and transport mechanisms. Although improvements were heavily patented, such was the competition that each manufacturer had perforce to implement each functional improvement on its own machines; great skill was expended in avoiding patent infringement, or at least achieving a gentlemanly agreement. The accumulative effect of these improvements followed much the same pattern as in any mature technology (such as the motor car for example): while the machines of the late 1930s bore a strong outward resemblance to the machines of the early 1920s, beneath the covers every functional subassembly had undergone successive cycles of optimisation and refinement. In addition to these internal improvements new ancillary machines such as the multiplying punch, the interpreter and the collator were introduced.

4 War and postwar (1940–1959)

At the time of the outbreak of World War II the two British punched-card machine manufacturers were among the largest and best equipped precision engineering concerns in the country. It was therefore inevitable that both

§In 1929 the selling company of the British Powers organisation became known as Powers-Samas. The name Samas derived from the French selling organisation.

BTM and Powers-Samas would be heavily involved in armaments production. In both companies a high proportion of productive capacity was used in the manufacture of aeroplane and gun parts, bomb sights and fuses, and fire control apparatus. Demand for all forms of office machines 'rose by leaps and bounds as administrative work and the collection of statistics expanded in the Services, in industry and in Government departments'[14]. Punched-card machine production thus continued alongside armament manufacture, distribution being supervised by the Board of Trade Directorate of Office Machinery.

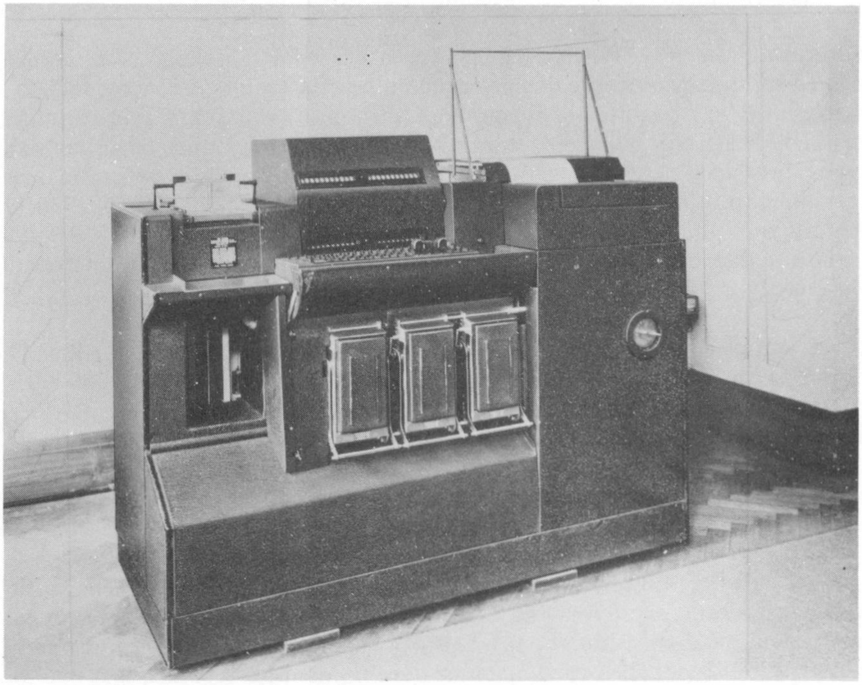


Fig. 4 BTM alphabetical rolling total tabulator, *c.* 1936

The technical staff of both companies were involved in military development work. In view of the current interest in the cryptographic operation at Bletchley Park, perhaps the most spectacular wartime R & D activity was BTM's involvement in what was known as 'Project Cantab'. This project was concerned with the construction of electromechanical code-breaking machines ('bombes') to decrypt the 'Enigma' traffic produced by the German cipher machines[15]. It is said that it was for this work that Ralegh Phillpotts received his knighthood in 1946, and for which H.H. Keen, F.V. Freeborn and C.G. Holland-Martin received OBEs (Phillpotts, incidentally, was not unaware of the irony that he received a higher honour than those who did the technical work!)*.

Powers-Samas also made a major contribution to wartime research, devoting many of its resources to the Vickers Mark XIV Computor [*sic*] for bomb-aiming:

> 'The success of the Mark XIV Instrument, in the development of which we were able to assist the Air Ministry, led to an extremely urgent demand for a number of pre-production models. Some 40 computers were produced in the Research Department together with several special machines for use in the production of special components'. (Minutes of Development Committee Meeting 143, 8th July 1943.)

Soon after the war, Powers-Samas was acquired by Vickers as a wholly owned subsidiary. Because of war-related research, basic R & D on punched-card machinery was drastically cut back during the war years. For example, in the Powers company the years 1940–1944 saw only nine new machine types produced and five minor improvements, compared with 29 new machines and 19 improvements in the previous five-year period 1935–1936[16]. Although on the cessation of hostilities both companies revived their R & D activities, they also found themselves in a seller's market of pent-up demand that partly masked the need for product innovation.

It was not perhaps until 1947 or 1948 that a real sense of urgency for R & D began to manifest itself. At about this time the R & D activity of both companies was put under additional strain by the termination of their licensing agreements with their American parents. In retrospect these were to be more crucial events than either company could have realised at the time because R & D costs were about to escalate out of all proportion to prewar experience.

In the case of Powers-Samas, its reciprocal licensing agreement with Remington-Rand expired in May 1950 and the American company declined to renew it. The British company had of course not made a proportionate contribution to punched-card machine development for several years due to other war-time priorities, and Remington-Rand took the somewhat opportunistic decision to sever its relationship with the war-weakened British company with the aim of competing in the British territories in which the latter had so long had exclusive rights.

BTM terminated its agreement with IBM by mutual consent in 1949. In fact the licensing arrangement with IBM had been an obligation of the most onerous kind to BTM throughout its existence. The agreement was that BTM paid IBM 25% after tax of its net revenues in exchange for the use of IBM patents: the outcome was that IBM actually made more profit from

*At the time of writing, the author had traced some of the people involved in this work, now mostly in their seventies. A detailed account will appear in another place.

BTM than did its shareholders. Furthermore, IBM's insistence that BTM lease rather than sell machines had hindered company growth due to lack of capital to finance the leasing. BTM was of course only too well aware that by severing its relations with IBM it would not only looe access to future developments, but it would also be exposed to IBM competition in Britain and the Commonwealth. However the company view was that:

> 'The firrancial benefits immediately accruing to the Company in the new circumstances will be devoted to strengthening the business in every direction necessary to meet competition. The Management ... will welcome this opportunity of proving that British effort and British skill can be matched successfully against any competitor in our business, whether national or international'[17].

The first R & D priority of both companies was to develop their standard punched-card machines (which were essentially unchanged from those sold in the late 1930s) to meet the competition that would be unleashed upon them once their American parents had built selling organisations in British territories. The awesome size of this R & D challenge was brought home when IBM announced in quick succession during 1948 and 1949 a daunting series of new products: the model 407 accounting machine, the 603 electronic multiplier and the 604 calculating punch. In every case these machines completely outclassed the offerings of the British companies. Remington-Rand were likewise offering electronic punched-card machines, which probably seemed more of a threat at the time than the fact that in 1949 it had acquired the Eckert-Mauchly Computer Corporation (UNIVAC), one of the brightest new computer development companies in the USA.

It was against this background that Powers-Samas more than doubled its 1949 R & D expenditure of £50 000 to a projected £139 000 for 1950; this represented a rise from about 2% to 5% of its revenue. By good fortune we have a remarkably clear picture of postwar research management in Powers-Samas by virtue of the large volume of documentation that has survived: there is an almost complete set of agenda, minutes and supporting papers of the various research committees for the period 1945–1959. The bulk of Powers-Samas development activity continued to be on traditional punched-card machines, to meet the pressing need for modern competitive machines – particularly an updated tabulator. In 1950 the company also established an independent computer laboratory at the Crayford plant of Vickers, with the somewhat conservative aim of introducing electronics 'as and when appropriate' into its punched-card machines.

This rather cautious approach to introducing electronic products was a cause for some concern outside the company. The National Research & Development Corporation (NRDC) took the initiative of establishing an Advisory Panel on Electronic Computers which held a single meeting on the 14th December 1949. The meeting was attended by both British punched-card machine manufacturers and by the majority of British electronics manufacturers:

'The outcome of the Advisory Panel meeting was that both the electronics manufacturers and the punched card machine manufacturers respectively represented that they were individually in positions to tackle the problems of an electronic computer development project as well as, for example, the International Business Machines Corporation in the United States. It was pointed out to the punched card machine manufacturers that, in the opinion of the Corporation, they had inadequate electronic staff and resources. It was apparent also that the manufacturers were not willing that the Corporation should take the initiative in launching a development project but agreed that the Corporation could usefully coordinate activities'[18].

In fact this concern at the lack of electronics capability of the punched-card machine manufacturers underestimated their resourcefulness. Within Powers-Samas, for example, major research projects were soon to include an electronic multiplying punch, a new super-tabulator and an electronic computer. Work began on the electronic multiplying punch (EMP) in 1950 and with a decisiveness that must have surprised outsiders a product, which sold in hundreds, was on the market by 1953. The new tabulator – marketed as the Samastronic in spite of its being entirely electromechanical – was an ambitious development based around a 300 lines per minute printer of novel design. Unfortunately the development was beset by many problems; machines were delivered many months late and were unreliable in service†. The programme-controlled computer (PCC) fared only somewhat better than the Samastronic, but it used the latter's printing mechanism and had its own problems of reliability. The Samastronic and the PCC both afford particularly clear examples of a technology in transition. The Samastronic (Fig. 5), whose functional specification was essentially that of a mid-1940s tabulator, was an example of a technology extrapolated beyond its practical limit. On the other hand, the PCC straddled the gulf between a calculating punch and a true stored-program computer. At the time of the merger with BTM in 1959 a fullscale computer PLUTO was under development in collaboration with Ferranti.

The present knowledge of the postwar R & D activity in BTM is somewhat limited as very little documentation appears to have survived. Fortunately, however, the majority of participants are still with us (and many are still with ICL) so that it will be possible to draw on 'oral history'. Like Powers-Samas, BTM at first saw electronics mainly in terms of enhancing its punched-card machines, and few resources were diverted to electronic computers as such, the major R & D resources going into such developments such as the 900 series tabulators. However, once the company seriously took to computer

†In 1959, when BTM and Powers-Samas merged to form ICT, the Samastronic proved to be a serious financial and marketing embarrassment that haunted the new company during its early years. A senior technical manager within ICL comments 'the Samastronic failed through straight bad mechanical engineering, a supreme irony for such an experienced organisation ... The Samastronic story is central to the history of punched card R & D in the UK. The real facts have never been published'.

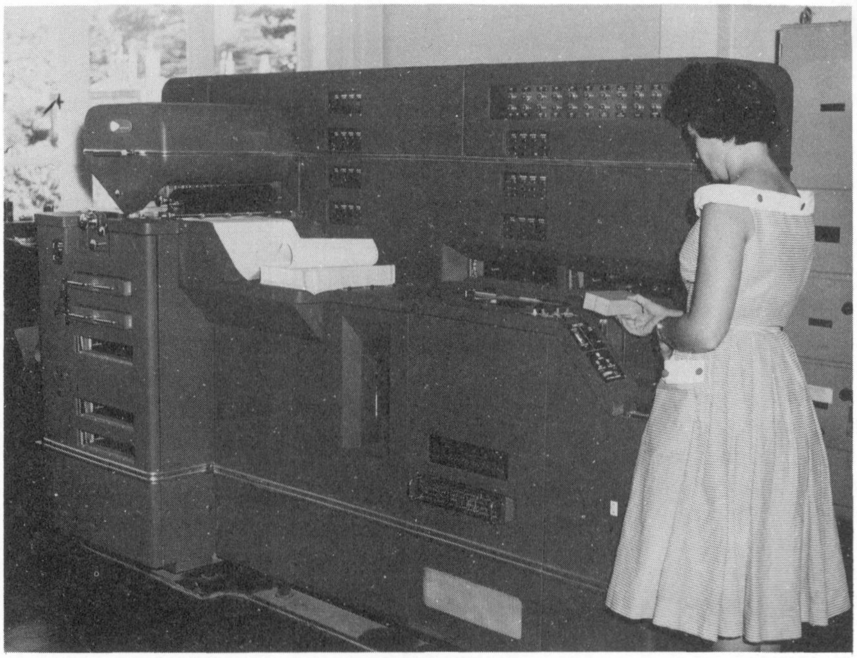


Fig. 5 Powers-Samas Samastronic tabulator, *c.* 1956

development it did so more decisively than did Powers-Samas, and there were three fullscale computer developments during the 1950s. A notable feature of all these developments is the willingness that the company showed in importing knowhow from outside at an early date.

In 1950 BTM appointed J.R. Womersley, then Superintendant of the National Physical Laboratory Mathematics Division, to head computer development. An experimental computer known as the HEC (for Hollerith Electronic Computer) based on a design by A.D. Booth of Birkbeck College was completed at Letchworth in 1950. From this development a production version the HEC 2M (also known as the model 1200) was sold from 1955. This machine was a scientific computer and only a few were sold, but from it derived the HEC 4, Britain's first and most successful first-generation commercial computer. This machine sold in two versions (the 1201 and the 1202), and well over 100 were made.

The second major computer development in BTM occurred in collaboration with GEC, with which it formed a jointly owned subsidiary company Computer Developments Ltd. in 1956. The pooling of BTM's data-processing resources and GEC's electronics skills led to the development of a small second-generation transistorised computer, the 'P3'. The P3 was rather slow in development and did not reach product status until 1960, as the model 1301. A third, and little known, development occurred in collaboration with the Laboratory For Electronics Inc. of Boston for the joint development of a new commercial computer, the APOLLO, and a large capacity drumstore; neither of these projects developed into BTM products. At the time of the merger in 1959 another major in-house development, the 'balanced data-processing computer' was under way and provisionally designated the model 1400: but the machine was based on valve technology and had to be abandoned.

In January 1959, after many months of negotiation and preparations for integration, BTM and Powers-Samas formally merged to become International Computers & Tabulators Ltd. At the time of the merger neither company was marketing a second-generation commercial computer, a most notable deficiency in the face of the very successful IBM 1401 to be launched later in the year. The company began to market its small second-generation 1301 computer in 1960 and from 1963 the much larger 1500 computer was made under licence from RCA. It was not until the launch of the third-generation 1900 series, however, that ICT had a machine that was something like the equal of its American competitors. The 1900 series had an unprecedentedly large development budget, partly government financed, and opened a new chapter in company R & D. A future paper will describe in detail R & D activity on computer developments from 1960 to the late 1970s.

References

The bibliography for the ICL company history, when it is complete, will contain several hundred items. The references below are intended to be representative of the primary and secondary sources, printed and 'near-print', on which the history will be based. The minute books of the companies from which ICL is descended are held in the ICL Secretariat Archives in Bridge House, Putney, where a large volume of legal and committee documentation is also located; these sources, though used, have not been cited in the text. Full runs of the company magazines *Tabacus. The Tabulator*, *Powers-Samas Magazine*, *Powers-Samas Gazette* and *ICT House Magazine* are held by the ICL Historical Collection in Stevenage. The Stevenage archive also holds several thousand brochures and technical manuals.

1 KOON, S.G.: 'Hollerith tabulating machinery in the business office', *Machinery*, 1913, **20**, 25–26.
2 AUSTRIAN, G.: '*Herman Hollerith: forgotten giant of information processing*', Columbia University Press, New York, 1982.
3 EVERARD GREENE, C.A.: '*The Beginnings*'. BTM, London, 1959.
4 TRUESDELL, L.E.: '*The development of punched card tabulation in the Bureau of the Census 1890–1940*', GPO, Washington DC, 1965.
5 CONNOLLY, J.: '*History of computing in Europe*', IBM World Trade Corp., 1967, 117.
6 DAVIS, M.D.: 'James Powers', National Personnel Records Center, Missouri, 11 August 1969.
7 BELDEN, T.G. and BELDEN, M.R.: '*The lengthening shadow: the life of Thomas J. Watson*', Little, Brown & Co., Boston, 1962.
8 BROUGHAM, L.E.: '*Report on visit to USA October 1929*'.
9 IBM: 'The light he leaves behind' [Obituary of J.W. Bryce], *Think*, April 1949, 5–6, 30–31.
10 EAMES, C. and EAMES, R.: '*A computer perspective*', Harvard University Press, Cambridge, Massachusetts, 1973.
11 Anon.: 'The Tabulator', *The Engineer*, 17 March 1911. 279–280.
12 Anon.: 'The first alphabetical printing unit', *Powers-Samas Gazette*, March/April 1956, 8–9.
13 ENGLISH, F.G.S.: 'The measure of progress', *Powers-Samas Gazette*, November 1957, 2–5.

14 HARGREAVES, E.L. and GOWING, M.M.: '*History of the second world war: civil industry and trade*', HMSO and Longmans Green, London, 1952.
15 HODGES, A.: '*Alan Turing: the enigma*', Burnett Books, London, 1983.
16 THOMAS, A. T. (An anonymous untitled undated typescript, probably writtèn by Arthur Thomas in 1952 at about the time of his retirement, describing machine developments 1921–1952; an appendix gives a chronological record of developments.)
17 BTM 'Statement by the Board of Directors', *Tabacus*, October 1949, 2.
18 NRDC,: 'National research and development corporation: computer project', NRDC Paper 132, February 1957.

Innovation in computational architecture and design

M.D. Godfrey
ICL Head of Research, Bracknell, Berks.

Abstract

This paper presents some of the motivation for innovation in computational architecture and design, and discusses several architectural and design ideas in the framework of this motivation. It is argued that VLSI technology and application architectures are key motivating factors.

Because of its unusual properties with respect to VLSI and application efficiency in certain areas, the ICL Distributed Array Processor is discussed in detail.

1.0 Introduction

The purpose of this note is to discuss some of the motivation for innovation in computational architecture, and, in this context, to review a number of computational architectures and describe an *active memory* architecture which is the basis of the ICL Distributed Array Processor (DAP)* products. Any discussion of new computational architecture must take account of the pervasive impact of VLSI systems technology. It will be argued that a key attribute of VLSI as an implementation medium is its mutability. Using VLSI it is natural to implement directly the computation of specific problems. This fact will induce a fundamental change in the structure of the information industry. While many new computational architectures do not fit well with this VLSI systems driven view of the future, the DAP structure, viewed as an active memory technology, may prove to be effective for the composition of important classes of application systems.

At present, there is a very high level of activity directed toward 'new' architecture definition. Much of this work has a negative motivation in the sense that it is based on the observation that since it has become increasingly hard to make 'conventional' architecture machines operate faster, one should build a 'non-conventional' (or non von Neumann) machine. This negative motivation has been pretty unhealthy, but it is good to keep it in mind as it

*DAP is a trademark of ICL

helps to explain much current work which otherwise would not have an obvious motivation. In my view, good architecture must make the best use of available technology in an essentially market-driven framework, i.e. form follows function. This was obviously true of the approach taken by von Neumann in defining the present 'conventional' architecture. And, it may help to explain why this architecture continues to be the dominant computational structure in use today.

Architecture can be thought of at various levels. It has been usual to distinguish between system, hardware, and software architectures. By implication, this note argues that the most useful context for architectural thinking is *application architecture.* Demand for higher efficiency will continue to decrease the prominence of conventional software. When full use is made of the technology of VLSI systems, the dominant mode of architecture and design expression will be integrated systems which efficiently compute results for a given application. In order to achieve this integration it will be necessary to define the basic structure of software in a manner that is consistent with the computational behavior of digital logic. These premises are not elaborated further in this note, but they are used to draw conclusions concerning the likely usefulness of the architectures that are discussed.

A related subject which is also not explicitly discussed below is that of *safety.* Current computational systems are unsafe in the specific sense that they often fail when used in a manner that the customer was led to believe was reasonable. The impending demand for demonstrably safe application systems is one of the key longterm directional forces in computation. The facts that conventional software is not based on a physically realizable computational model, and that it is not subject to reasonable test will both work against its continued use. Safe computational systems will be built based on a model of the behaviour of digital logic such that the domain of proper use and the expected behavior within that domain can be specified and demonstrated in a convincing manner. While in some cases a convincing demonstration can be by example use in 'typical' situations, in other cases it will be necessary to assure proper behavior without the time and cost associated with extensive practical trials. It is the later cases that demand a wholly new formulation. For a precise statement of the limits of present software technology see [7].

2.0 Architecture and computational work

2.1 The current architectural scene

A few key factors should dominate architectural thinking:

1 The time it takes to communicate information along an electrical conductor imposes a strict limit on the speed of individual computational elements [4, Chapter 9].
2 The advent of VLSI technology has fundamentally changed the

technology constraints. VLSI permits the composition of highly complex two dimensional structures in a uniform physical medium. The current state of VLSI fabrication technology permits about 1/2 million transistors on a single chip. Projections indicate that densities of 20 million transistors are theoretically feasible, keeping the size of the chip constant. This means that a very high level of architectural definition must take place in the context of the base material from which the system will be constructed. The two dimensional nature and electrical power considerations lead to the observation that VLSI is a highly concurrent medium, i.e. it is likely to be more efficient if many of the individual elements on a chip are doing something at the same time. Communications is a dominant cost. Communication costs increase by a large increment when it is necessary to go off-chip. Thus, efficiency is improved if the number and length of communication paths is minimized, and the bulk of communications is localized within any chip.

3 Historical evidence indicates that the total demand for processing power is essentially unbounded. Thus, an increase in perceived computational performance, at a given price, will result in a very large increase in demand. This effect really does appear to have no fixed limit. Within this demand behavior there is a key discontinuity which is referred to as the interaction barrier. A qualitative change takes place when a computational task can be performed in a time that is within a human attention span.

4 Digital logic must be designed and implemented to operate correctly. There is no tradeoff between speed and correctness or safety. The tradeoff is between speed and efficiency of computational work. Higher performance, for a given technology, requires more energy and more space. The understanding of the locus of efficient points in the space-time-energy domain is an unsolved problem of fundamental importance.

2.2 Computational work and performance measures

It is common practice to describe the performance of a computational system in terms of the rate at which instructions, or particular classes of instructions (operations) are carried out. This is the basis of the MIPS (Million Instructions Per Second) and MFLOPS (Million FLoating-point Operations Per Second) measurement units. The basic element of computational work is the determined rearrangement of data. Thus, the appropriate measure of performance should be a measure of the rate at which a determined rearrangement can be carried out. Such a measure must evaluate the rate at which the required rearrangement can be decided and the rate at which the data items can be transformed into the required arrangement. In many computations the decision time and complexity dominate the data arrangement time. An extreme case of this kind is sorting. If the required record order was known at the start of a sort, the resulting sort time would be quite short. These observations suggest that the MFLOPS measure may be misleading as it tends to neglect the decision work that is required in all useful computations.

The remainder of this note is in three main Sections. First, we will discuss some of the main development efforts which are known to be underway. Then, we will review the active memory architecture as embodied in the DAP and indicate its context for comparison. For the present purposes the key feature of the DAP structure is that it is a component technology which may be composed into specific systems and which thus may be effective in a VLSI systems design framework. Finally, we will briefly discuss the expected future direction of VLSI-based architecture and design.

3.0 Alternative architectures

Not only has there been considerable recent discussion about new architectures, but there has been increasing discussion about the terminology and taxonomy of these architectures. All this is still pretty confused. I will try to keep things simple, and avoid as much of the confusion as possible. Approximately, the architectures will be described in order of increasing specialization, but this is only very rough as the notion of specialization is itself not simple.

A standard taxonomy uses the following notation:

SISD–	Single Instruction, Single Data stream: a single conventional (von Neumann) processor,
SIMD–	Single Instruction, Multiple Data: a set of processing elements each of which operates on its own data, but such that a single stream of instructions is broadcast to all processors,
MIMD–	Multiple Instruction, Multiple Data: typically, a collection of conventional processors with some means of communication and some supervisory control.

The remaining possibility in this taxonomy, MISD, has not been much explored even though, with current technology, it has much to commend it.

In addition, the 'granularity' of the active elements in the system is often used for classification. A system based on simple processors which operate on small data fields is termed 'fine-grained', while a system of larger processors, such as 32 bit microprocessors, is termed 'coarse-grained'. This classification can tend to conceal other key distinctions, most prominently the nature and efficiency of communications between the active elements, and thus it may not improve useful understanding.

3.1 Multiprocessors (MIMD)

Multiprocessors, with the number of processors limited to about six, have been a part of mainframe computing for about 20 years. The idea has been rediscovered many times, most recently by designers of microprocessor-based systems. A pure 'tightly-coupled' multiprocessor is composed of several processors all of which address the same memory. Each processor runs its own independent instruction stream. A basic hardware interlock (semaphore) is required to control interaction and communication between the processors. Various software schemes have been developed to manage these systems. The most effective schemes treat the processors as a virtual resource so that the programmer can imagine that he has as many processors as he needs, while the system software schedules the real processors to satisfy the user-created tasks in some 'fair' manner. In many systems the software places restrictions on user access to the processors, either real or virtual. However, there is a fundamental hardware restriction which is imposed by the need to have a path from each processor to all of the memory. The bandwidth of the processor-memory connection is a limitation in all instruction processor designs, and the need to connect several processors just makes this worse. If a separate path is provided for each processor, the cost of the memory interface increases very rapidly with the number of processors. If a common bus scheme is used, the contention on the bus tends to cause frequent processor delays. The current folklore is that the maximum realistic number of processors is around six to eight. Thus, in the best case, this arrangement can improve total throughput of a system by around a factor of six. To the programmer, the system looks either exactly like a conventional system (the system software only uses the multiple processors to run multiple 'job-streams') or, in the most general software implementations, it looks like a large number of available virtual processors. However, in either case, total throughput is limited by the maximum realizable number of real processors.

Examples of such systems are the Sperry 1100 Series, Burroughs machines, Hewlett-Packard 9000, IBM 370 and 3000 Series, and ICL 2900, and various recent mini's and specialized systems.

This architecture is likely to continue to be used, particularly in dedicated systems which require high performance and high availability since the multiple and, in some cases, exchangeable processors can make the system more responsive, and more resilient to some kinds of failures.

3.2 The multiflow machine

One of the very few system or problem driven architectures is the 'multiflow' design which was developed at Yale University [5] and subsequently at Multiflow Inc., formed by the Yale developers. Their design is an integrated software–hardware design which attempts to determine the actual parallelism in an application (expressed without explicit regard for parallelism) and then to assign processors and memory access paths to the parallel flow paths. This is done by analysis of the program and sample input data. This use of data is the key distinctive feature of this system. The hardware is similar to conventional multiprocessor organization, as described above, except that the processor-memory interface is carefully designed so that the software can organize the parallel computation in a manner that minimizes memory contention.

This could result in a significant improvement over conventional multiprocessor techniques, but is unlikely to produce more than a factor of ten. To the user, this looks just like a conventional sequential system. If a sequential language is used for programming this system then the potential benefit of compact representation is lost in exchange for not having to recode existing programs.

3.3 Arrays of processors (MIMD)

This is the area that is getting lots of publicity and lots of DOD and NSF money. Projects at Columbia (non-von), Caltech (Cosmic Cube), and NYU are examples of this structure. The Caltech project has been taken up by Intel, and others such as NCUBE Inc.

The common thread in many of these designs is to arrange, in a more or less regular structure, a large set of conventional microprocessors each with its own memory. Thus, this design solves the problem of common-memory systems by having each processor have its own memory. However, this structure suffers from the problem of communication between the processors which is made more severe since they cannot communicate through shared memory. Since the communication scheme has to be determined once and for all when the machine is designed, it cannot be optimized for widely differing application requirements. The current unsolved problem in this structure is how to transform current problems, or create new problems, which match the connection and communication structure of the designed machine.

It is usual to talk about at least 64 processors, and some projects are planning for several thousand. Generally the number of processors is tied to the funding requirements of the project, rather than to any deduction from application requirements. The current choice of processor is variously Intel 286, Motorola 68020, INMOS transputer, etc.

There are several projects which use this general structure, but which attempt to organize the processing elements and their interconnections in order to provide faster operation of some forms of functional or logic programming, typically in the form of LISP or PROLOG. ICOT is building such a system and the Alvey Flagship project plans a similar effort: in this major project, the largest under the Alvey Directorate, ICL is leading a consortium in which the other partners are Plessey, Imperial College (London) and Manchester University. Thinking Machines Inc., spun off from MIT, seems to be furthest along on a VLSI implementation. Their machine, called the Connection Machine [6], is also distinctive, when compared to the class of systems mentioned so far, in that the processing elements are relatively simple single-bit processors. In this respect the machine can be described as 'DAP-like', but this analogy is not very close. In particular, the machine has an elaborate and programmable processor-to-processor communication scheme, but no direct means of non-local communication.

3.4 Vector processors

These designs differ from any of the previously described systems in that they introduce a new basic processing structure. The fundamental precept of these designs is to extend the power of the processor instruction set to operate on vectors as well as scalars. Thus, if an instruction requests an operation on a vector of length, say, 64 and the operation is carried out in parallel with respect to the elements of the vector, then 64 times as much work was done by that instruction. This is an example of a general argument that says: if there is a limit to the speed at which instructions can be processed then it may be better to make each instruction do more work. Seymour Cray thought of this approach, and the CDC and Cray Research machines which he designed are the best embodiments of the idea. Experience with three generations of these machines, particularly at the US National Research Labs (Livermore, Los Alamos), has led to the conclusion that it is quite hard work to arrange a given problem to match the vector structure of the machine. The best result is something like a factor of ten improvement over conventional techniques. These machines are inherently quite complex and therefore expensive, and performance suffers with attempts to reduce the cost.

Considerable effort has been put into compilers, particularly Fortran, which can automatically 'vectorize' a program which was written for a conventional machine. This work has not been very successful because the actual dimensionality of a problem is usually not indicated in the program. The dimensionality is only established when the program executes and reads in some data. This fact was understood by the Multiflow people.

The current design direction in this area is to try to combine vector processing and multiprocessor systems, since the limits of the vector extension have substantially been reached. This is leading to extremely complex systems.

3.5 Reduced instruction set designs (RISC)

These designs are motivated by the opposite view from that held by the vector processor folks. Namely, it is argued that a processor can execute very simple instructions sufficiently quickly so that the fact that each instruction does less work is more than offset by the high instruction processing rate. A 'pure' RISC machine executes each of its instructions in the same time, and without any hardware interlocks which would ensure that the results of the operation of an instruction have reached their destination before the results are used in the next instruction. This adds greatly to the simplicity of the control logic in the instruction execution cycle. However, it places the burden of ensuring timing correctness on the software. Generally, the RISC designers have concentrated on reducing the number and complexity of the instructions and, therefore, reducing the number of different data types on which the instructions operate. However, they have left the 'size' of the data items alone. Thus, RISC machines operate on typically 32 bit integers and, sometimes, 32 and 64 bit floating point numbers. Thus, they are much like conventional machines except that the actual machine instructions are reduced in number and complexity. By contrast, the DAP approach is to drastically reduce the allowed complexity of the data at the individual processor level, but to provide for direct operation on complex structures through the large number of processors. The instructions which operate the DAP PE's are, in current designs, very much simpler than in RISC designs.

In addition, but in no necessary way connected with the reduced instruction set, RISC machines have been defined to have large sets of registers which are accessed by a structured address mechanism. This construct is used to reduce the relative frequency of memory references, thus permitting fast instruction execution with less performance restriction due to the time required for access to data in the main memory.

One can argue that RISC and vector processors are in pretty direct conflict: one says smaller instructions are better while the other says bigger instructions are better. It would be nice if we had some theory which could shed light on this conflict. We do not. The empirical evidence tends to indicate that they both have it wrong: i.e. the 'biggness' of the instructions probably does not matter much.

The main research on RISC architecture was done originally at IBM Yorktown Heights, Berkeley, and Stanford. Both IBM and Hewlett-Packard have recently announced RISC-based products.

A basic premise of the RISC approach, as is true of the vector approach, is that the software, or something, can resolve the fact that the users' problems do not match well with the architecture of the machine. In the RISC case the language compilers and operating software must translate user constructs into a very large volume of very simple instructions. In some situations, such as error management, this may become rather painful.

3.6 Processor arrays (SIMD)

The DAP is often referred to as a SIMD machine, but this is another point at which the taxonomy of new architectures can become confusing and confused. The DAP is a single instruction, multiple data machine in the sense that a single instruction stream is broadcast to all the processors each of which then operates on its own data. However, the data object in a DAP is very different from the data object in most other SIMD, or other, machines. (This suggests that the xIyD classification, by putting the instruction character first has got the priority wrong: the data are what really matter.)

Several SIMD machines which operate on conventional data fields, such as 32 bit integer and 32 or 64 bit floating point, have been built. However, the current view in the 'big-machine' world is that MIMD should be better: e.g. Cray X-MP, HEP.

3.7 *Dataflow machines*

For many years thought has been given to the idea that computation should be driven by the data, not the instructions. One version of this thinking gave rise, in the late 1970's, to 'Dataflow Machines', particularly at Manchester University and MIT. The basic construct of Dataflow Machines is a system for managing 'instructions' which are composed of data items and the operation which is to be performed on the data items. Data items enter the system and when all the required items are available for a given instruction, the instruction is sent to the operation unit which performs the intended operation and produces the data result. This result may then complete some other instruction which was waiting in a queue. This instruction is then processed. It has been argued that this arrangement of data-driven scheduling can improve the parallelism of a computation since many instructions may be in progress at any time, and the work gets done as soon as the data become available. However, the selection of instructions for processing must be done serially and thus parallelism is not obviously improved over conventional designs. In addition, new language techniques are required to create programs for such machines.

It can be argued that the basic idea underlying dataflow machines is sound but that the recent research attempted to apply it at much too low a level.

4.0 The active memory architecture

The active memory structure of the Distributed Array Processor (DAP [2, Chapter 12]), first developed by ICL, will be described in a more complete way because it may be viewed as forming a basis for a distinctive capability which has not been developed in other systems. The DAP structure by its nature leads to the formulation of problems in terms of the required rearrangement of the data. Thus, its efficiency tends to be dependent on the spacial distribution of data-dependent elementary decisions. (A formal notation for efficient organization of the dynamics of data arrangement is presented in [1].)

Broadly, the DAP is an active memory mechanism such that an array of processing elements control the manipulation of data in the memory structure. The array of processing elements, each of which addresses a local memory, are operated by a single instruction stream and communicate with their nearest neighbours, in some topology.

Progressively more narrow definitions also include the restriction that the processors have some particular amount of local storage, that the processor width is one bit, and that particular structures are provided for data movement through the local store structure. For applications that require communication beyond local neighbours, it may be essential to have row and column data paths which allow the movement of a bit from any position to any other position in a (short) time which is independent of the distance moved.

While the above definitions are useful for some purposes, an external definition is more appropriate for understanding some applications and market opportunities. A useful external definition is: A DAP is a subsystem which is directly effective for execution of DAP-Fortran, or of a sub-set of the Fortran 8X array extensions. The term 'directly effective' is intended to mean that there is a close match between the language construct and the corresponding architectural feature, and that the resulting speed of operation is relatively high. The relevant Fortran extensions permit logical and arithmetic operations on arrays of objects. The DAP performs operations in parallel on individual fields defined over the array of objects.

4.1 Performance

So far no mention has been made of absolute performance. This is appropriate as it is assumed that a contemporary DAP will be constructed from contemporary technology. Therefore, the important question is what is DAP relatively good at? The definitions above are meant to make it clear that a DAP is relatively good at computations which involve a relatively high density of operations, including selection and conditional operations, on replicated structures and which require parallel rearrangements of data structures. The replication may be in terms of the dimensions of arrays, record structures, tables, or other patterns. For example, routing algorithms in 2 dimensions satisfy this requirement very nicely.

4.2 Cost

Cost is an important consideration in the definition of a DAP because if a DAP is defined as being relatively good at some computation, this must be taken to mean that it is relatively more cost-efficient. DAP costs are differentially affected by VLSI technology. The basic DAP structure scales exactly with the circuit density. This simple correspondence between DAP structure and VLSI structure is a useful feature which must be taken into account when projecting possible future cost-effective DAP structures. The main discontinuity occurs at the point where a useful integrated memory and processor array can be produced. Roughly, the technology to produce 1 megabit RAMs will permit such an integrated implementation.

To indicate present cost characteristics, 2 micron CMOS (2 layer metal) can support, approximately, an 8×8 DAP processor array. The chip fabrication cost is of the order of \$10. Thus, a 16 chip set to provide a 32×32 array would imply a chip cost of \$160. This structure would require external memory to compose a subsystem. Using emerging VLSI technology it will be possible to construct memory and processors on a single chip, thus improving performance and reducing the cost to approximately the cost of the memory.

4.3 Array size

It is reasonable at some levels to define a DAP without reference to the dimensions of the processor array. However, if one asks how well a DAP can solve a problem the array size becomes a prominent factor. For practical purposes it must appear to the user that the array has dimensions within the range of about 16 to 128. (Or, in other words, the array contains from 256 to 16384 processors.) With present techniques the user must arrange his data structures to match the DAP array dimensions. Most realized implementations of DAPs have used a square array structure. Whether the array is square is not very significant, and should not be a part of any definition. However, square arrays are obviously simpler to program and will likely continue to be the standard form. It is more significant that the dimensions should be a power of two. Many of the established techniques rely on composition based on this fact.

4.4 Processor width

The processor width is a key element of DAP structure. It can be argued that the (single bit) width of the processor should be a defining feature of a DAP. It is probably somewhat more realistic to state that a DAP must be capable of efficient operation in a mode that makes it appear to the user as if the processors were single bit wide. With present techniques this implies that the processor width must be quite narrow. A wider processor width might make an array system, such as the Caltech (Intel) Cosmic Cube, but it would definitely not be a DAP.

4.5 Local memory size

The size of the local memories, within limits, does not affect the definition of a DAP. However, the availability of substantial memory, so that the system can properly be viewed as a three dimensional memory with a plane of processors on one face, is an essential feature. The memory must be large enough to contain a substantial part of the information required for a given computation. The amount of memory associated with each processing element has an important effect on both performance and detailed programming. Typically, each processor may have 16 k bits of local memory, but greater memory size, as usual, permits efficient solution of larger or more complex problems. Particularly in VLSI technology, there is a direct tradeoff between array size and local memory size on a chip. How to best make this tradeoff is not well-understood.

4.6 I/O and memory-mapped interfaces

The interface of the array structure with the outside world is an important (for some applications, the most important) design feature. Increasingly, it will likely be necessary to construct interfaces to match specific application bandwidth and data ordering requirements. However, choices in this area do not substantially affect the DAPness of the array structure.

4.7 Language interface

For many purposes a useful definition of a DAP is in terms of the high-level language interface which it may support in an efficient manner. An essential characteristic of a high-level language for operation on a DAP is that it should raise the level of abstraction from individual data items to complete data structures. A consequence of this is that the parallelism inherent in the data is no longer obscured by code which refers to individual data items. This both permits expression of an algorithm in a more concise and natural form and causes the high-level language statements to correspond more closely to the operation of the DAP hardware.

Such a language interface may, of course, encompass a wider range of architectures than a specific DAP design. A language definition suitable for DAP should encompass other SIMD designs and also SISD systems.

5.0 VLSI-based architecture and design

VLSI is developing into a highly mutable *design* medium. This will diminish the need for general-purpose systems. Instead, it will become common practice to create the application implementation directly in VLSI. (For examples of this approach see [3] Part 2.) One way of viewing this change is that it raises system architecture to a set of logical constructs which guide the implementation of application designs. Much work remains to be done before a good working set of abstract architectural principles are available. In addition, standardized practices and interface definitions are required at both abstract and practical implementation levels in order that efficient composition can be realized in this form. However, the economic benefits of this mode of working will tend to ensure that the enabling concepts, standards, and supporting infrastructure will emerge relatively quickly.

5.1 Application-Specific Processors

VLSI technology has already caused an increased interest in application specific processing elements. This trend is likely to continue as the costs of design and reproduction of VLSI subsystems continue to fall relative to the cost of other system components. The most obvious examples of such processing elements are the geometry engines in the Silicon Graphics workstations, and the various dedicated interface controller chips for Ethernet, SCSI, etc.

Designs have also been produced for such things as a routing chip. This suggests that some such special architectures may be close to the DAP. The designs that are close to the DAP share a fundamental DAP characteristic: the optimal arrangement of data in memory is key to efficient processing.

5.2 Application Specific Subsystems

It is easy, in principle, to generalize the notion of application specific processors to application specific subsystems, such as signal processing, vision, or robotic subsystems. Again, VLSI continues to make such specialization increasingly attractive. The performance benefits of dedicated logic increase with the level at which the dedicated function is defined. With present VLSI technology it is possible to build chip sets which, for example, solve systems of non-linear difference equations at a rate about ten times the rate possible by means of programming a machine such as a Cray [3, Chapter 13]. The efficiency gain comes, in large part, from the fact that the specialization permits many decisions to be incorporated into the design. Specifically, none of the costs associated with the interpretation of a sequence of instructions exist at all. An additional benefit of such subsystems is that they require no software.

The skills that are required to design and implement such a dedicated solution to a real application problem are not now widespread, and supporting tools and techniques are underdeveloped. However, this general approach will dominate efficient computation in the long run.

6.0 Conclusion

In the long-run the efficiency of direct implementation of specific computations in silicon will dominate other techniques. However, before this level of efficiency can be achieved in a routine manner a number of research problems must be solved and substantial new infrastructure must be established. In addition to the need for improvement in VLSI design methods, a new level of understanding and definition of software will be required.

7.0 Acknowledgments

Several people have made very helpful comments on previous drafts of this paper. Steve McQueen, Peter Flanders, and David Hunt made particularly detailed and useful comments. As always, remaining faults rest with me.

8.0 References

1 FLANDERS, P.M., 'A Unified Approach to a Class of Data Movements on an Array Processor', *IEEE Tr. on Comp.* Vol. C-31, no. 9, Sept 1982.
2 ILIFFE, J.K., *Advanced Computer Design*, Prentice-Hall, 1982.
3 DENYER, P. and RENSHAW, D., *VLSI Signal Processing: A Bit-Serial Approach*, Addison-Wesley, 1985.
4 MEAD, C. and CONWAY, L., *An Introduction to VLSI Systems*, Addison-Wesley, 1980.

5 RUTTENBERG, J.C. and FISHER, J.A., 'Lifting the Restriction of Aggregate Data Motion in Parallel Processing', IEEE International Workshop on Computer System Organization, New Orleans, LA, USA, 29–31 March 1983 (NY, USA, IEEE 1983) pp 211–215.
6 HILLIS, W.D., *The Connection Machine*, MIT Press, 1985.
7 PARNAS, D.L., 'Software Aspects of Strategic Defense Systems', *American Scientist*, vol. 73, No. 5, pp. 432–440, 1985, and reprinted in CACM, Vol. 28, No. 12, pp. 1326–1335, Dec. 1985.

REMIT: A natural language paraphraser for relational query expressions

B.G.T. Lowden and A.N. De Roeck
Department of Computer Science, University of Essex, England

Abstract

The aim of this paper is to give an overview of the work carried out by Essex University, under ICL grant UEI, on the design and development of a formal query language to natural language interpreter to aid query verification in a relational database environment. The REMIT system – Relational Model Interpreter and Translator – has been developed to work in conjunction with the ICL natural language enquiry interface, NEL, developed as a research project to translate English query expressions into the formal query language Querymaster.

1 Introduction

Of the many problems facing the casual user of a database enquiry system probably the most difficult is gaining a competent understanding of the associated query language. Given that he manages to construct a well-formed query expression, there is no guarantee that it exactly reflects the original question. In a study of Query by Example (Thomas and Gould, 1975), it was found that 27% of the queries analysed were syntactically correct but gave the wrong answer.

Natural language processing is seen by some as the most promising solution to these difficulties. N.L. Interfaces, however, can create problems of their own. They cannot counter the users' often inflated ideas about what is a reasonable question to ask. Furthermore, natural languages are typically ambiguous and, although transparent to the querent, can be opaque to the machine. The interpretation placed on these queries, by the system, may therefore itself be ambiguous or otherwise misleading.

One approach to improving this situation is to offer the user a paraphrase of what the system has taken his question to mean. He can then verify whether the interpretation of his question corresponds to what he intended or, in the case of ambiguous input, select the alternative that does.

The remainder of this paper describes such a paraphraser designed at the University of Essex and implemented in Prolog. The system generates

English paraphrases for questions interpreted by the database query system NEL, (West, 1985) a research project at ICL, formerly known as QPROC (Wallace and West, 1983). NEL so far comprises a NL front end which maps natural language text onto expressions in the formal query language Querymaster (ICL, 1983, 1985), which are then used to retrieve data, for example from the database SCOPE (Ref. 13).

The paraphraser has also been designed to deliver paraphrases for queries directly formulated in Querymaster. As a consequence, unlike most NL feedback systems, it can also help users who do not have access to NL input facilities but must use a formal language. Furthermore, because the paraphraser assumes an extended relational calculus as an underlying representation, it can with little extra effort be modified to work from most query languages currently available.

2 Design approach

A paraphraser can be seen as a mechanism that provides a mapping between an underlying formal representation and a natural language text. Considering our task of providing a casual database enquirer with a useful paraphrase of his question, the representation selected must reflect the understanding the system has of the users' original input. This understanding must then be translated into clear, unambiguous and grammatical textual output.

2.1 Selection of the underlying representation

Paraphrasers can be classified into two groups, according to the nature of the underlying representations they assume. One type is designed specifically to operate alongside a NL front end (McKeown, 1979). The user introduces his question in NL, which is then parsed into a structure making linguistic facts explicit about the input. This linguistically motivated representation then serves as a basis for the paraphrase. The mapping between the formal representation and text established by paraphrasers of this kind can be said to be 'close' since most of the linguistic information the synthesiser may need is readily available.

On the other hand, not all questions a user may formulate, in natural language can be evaluated against a database. It is important, therefore, that the NL query is mapped into a formalism reflecting the limitations of the Database Management System before it is paraphrased, otherwise the result may be a paraphrase of a question the system cannot ultimately handle. Also paraphrasers working from linguistically motivated representations cannot work independently from a parser that will build the necessary structure. They do not help the user who has no access to a NL front end and attempts the use of a formal query language.

Another class of paraphraser works from representations which capture exactly that information which can be evaluated against a database, usually a parse tree of a formal query. If used with an NL front end they can report only on relevant ambiguities in the input text so far as they correspond to alternative formal queries. However, since these representations are linguistically under-specified, the mapping between them and NL text is 'distant' and more difficult to establish. The linguistic facts which characterise the resulting paraphrase must be decided upon without reference to any information which a linguistically motivated parse of an equivalent question might provide. Although this approach is more constrained than its alternative it does ensure that the paraphraser can be used both with, and independently of, any NL front end available and, as such, is the one adopted for this project.

2.2 Portability

The preceding discussion would appear to suggest Querymaster as the obvious candidate for the underlying representation. The choice of Querymaster expressions as the input for the paraphraser would, however, restrict the paraphraser's portability to those Database Management Systems capable of supporting that query language.

In order to retain all the advantages of a paraphraser working from representations which capture exactly the information present in a formal query language expression and, at the same time, to increase its portability, the mapping process between formal query language expressions and NL text has been split into two stages using an intermediary formalism. The choice of that formalism has been guided by two considerations. Firstly, it must be capable of expressing exactly the information that can be captured by any Querymaster expression. Secondly, it must be possible to identify an exact mapping from any query language into an expression of that formalism.

These two requirements are satisfied by the use of an applied relational calculus as an intermediary representation. Codd's relational calculus is well defined and relationally complete (Codd, 1971, 1972). When extended by a range of library functions (Date, 1977), it has at least the retrieval power of most query languages currently available. Furthermore, it can be shown (Ullman, 1980) that an exact mapping exists between an expression in a relationally complete language and an expression in the relational calculus (and vice versa), provided that it defines a derivable relation.

Part of the project, therefore, concerned the development and implementation (in Prolog) of a transducer which maps Querymaster statements into the relational calculus (Shephard, 1985). Its function is independent of the main body of the paraphraser and further reference to the latter will assume that it operates directly from the relational calculus.

This modular concept means that the process of adapting the paraphraser to work from other relational query languages is relatively straightforward.

2.3 Grammaticality

It is clearly important that the text produced by the paraphraser is well-formed with respect to the grammar rules of the human language used to describe the query (in this case English). In general, grammaticality is best ensured by making reference to a linguistic theoretical framework within which such rules can be formulated. Since many such frameworks exist some guidelines for the choice were drawn up. First of all, the framework should be implementable. This means that it should be formally specified to a level where an equivalent program can be written. Secondly, the syntax of the RC bears no resemblance to the syntax of English. The mapping between RC formulae and English texts can thus be called 'distant' and as far as possible left to the linguistic part of the implementation. This means that that theory which allows for the syntactically least specified underlying framework will be preferred.

For these reasons the choice made was Lexical Functional Grammar (Kaplan & Bresnan, 1983). It is a generative linguistic theory allowing for the specification of grammar rules for English. LFG has the advantage of being highly implementable – in fact it was designed from a computational linguistic point of view. It defines a mapping between sentences of English and their underlying, syntactically poorly specified, predicate/argument structures. It has an additional advantage in that mapping is stratified, using several intermediary representations, each offering an indication of which linguistic information needs to be specified at a given stage in the process.

2.4 Non-ambiguity

Whereas grammaticality can be guaranteed by reference to a grammar, non-ambiguity cannot. The problem arises from the fact that all human languages are ambiguous. Grammars account for ambiguity but do not seek to avoid it. In short, if a paraphrase is characterised only as a grammatical text, of some human language, then it follows that it is potentially ambiguous.

Ambiguity is a phenomenon that is difficult to control. No measure for a degree of ambiguity exists. One may attempt to parse the output text and thus try to gain some such measure, but many different sorts of ambiguity occur and it is not clear whether any grammar can account for all of them. Lexical ambiguity in particular is problematic and, in the extreme case, words may mean a variety of different things to different users. This is often dependent upon the user's background, and totally beyond the control of any grammar formalism. The solution adopted by REMIT was to concentrate on ambiguities which MUST be avoided in the paraphrases at all cost in order to preserve meaning.

Since the aim of a paraphrase is to query verification, the ambiguities which must be avoided are those significant with respect to query evaluation. These mainly relate to the scope of logical connectives and quantifiers occurring in an expression of the unambiguous, formal query language. Consider, for example, the following sequence of logical conjunctions and disjunctions:

$$a \wedge (b \wedge (c \vee (d \wedge e)))$$

'a and b and c or d and e' is not an adequate rendering of the above bracketed expression since all indication of scope is lost. We found that the written form of a human language, stripped of expressive devices such as intonation and stress patterns, is extremely ill-suited to express scope relationships of this type. If attempts are made to describe the scope information by means of punctuation and special words (e.g. either, both, all three of the following, ...) then the resulting linear text becomes illegible and unhelpful as a paraphrase. REMIT solved this problem by abandoning the idea of a paraphrase as a linear text and by adopting the view that scope is best represented hierarchically. The paraphraser retains some degree of bracketing in the output text which is then used to display the result on the screen using indentation as a means of conveying scope. The formal sequence above would therefore, for REMIT, result in a paraphrase of the following format:

```
    a
and b
and either c
        or d and e
```

2.5 Readability

The requirement that 'readable' paraphrases must be delivered has in part been catered for by the solution adopted to avoid ambiguity. A side effect of structuring the output text has been that it becomes indeed more readable and thus more friendly as vital scoping information is passed on visually to the user.

However, there is more to 'readability' than producing a grammatical text and displaying it in a particular format. The text must also be coherent, not just syntactically, but conceptually. What we mean by this is explained in more detail in the next section.

3 A model for the SCOPE database

Paraphrasing expressions in a query language comes down to selecting and organising the appropriate lexical material for describing, in a human language, what that query stands for in terms of the information that must be retrieved. Query languages and the RC are formal languages, i.e. their semantics with respect to retrieval is defined unambiguously on the basis of their syntax. As a consequence, the syntactic structure of a formal expression can be viewed as a shorthand for what it 'means' and can be used in order to guide the paraphrasing process.

Nevertheless, paraphrasing expressions of these formal languages can be problematic. Their syntax bears no resemblance to the syntax of a human language and a paraphraser must thus establish more than a simple syntactic transduction. Also, these formal expressions are poor in conceptual information about the domain or field a particular database covers. Although one can produce literal paraphrases relying solely on the information present in a formal query, the result will be a stunted incoherent rephrasing of the formal expression. In general, human language text is rich in conceptual information. If a paraphraser is to deliver texts that are acceptable and helpful to naive users then it must be able to incorporate conceptual information in its output.

This conceptual information cannot be collected from the database itself. Databases are implementations of formal objects that allow for storing and manipulating large bodies of knowledge. Although the administrative organisation of a relational database will often, to a large extent, be compatible with the conceptual structure of the field it covers, this is largely due to the 'common sense' of database engineers and such an organisational correspondence cannot always be guaranteed.

In addition formal query languages are totally devoid of any such conceptual information. Consider, for instance, the following RC formula:

{(CUSTOMER.ADDRESS, WAREHOUSE.CODE):True}

as might be expressed over the example SCOPE database, reproduced in Fig. 1 from (ICL, 1983). This is a perfectly well-formed RC expression. It has an equivalent in all relational query languages and the result will be the Cartesian product of all customer addresses with all warehouse codes. Although this is certainly a legal query, it is hard to see what it might 'mean' in conceptual terms and why anybody might want to formulate it. Paraphrasing queries of this kind is extremely difficult, even for people, short of saying 'Give me the Cartesian product of all values for...'

Still, most questions which users care to ask do make sense and usually carry conceptual content. They centre around a focal point or FOCUS which is not explicitly marked as such in the original formal expression, but which can be derived from it given conceptual information about the field the database covers.

This conceptual information will be contained within a MODEL of the database in question. Such a model must be constructed for any database with which the paraphrase will operate, and provides the linguistic/conceptual information which is necessary for the delivery of coherent and elegant paraphrases.

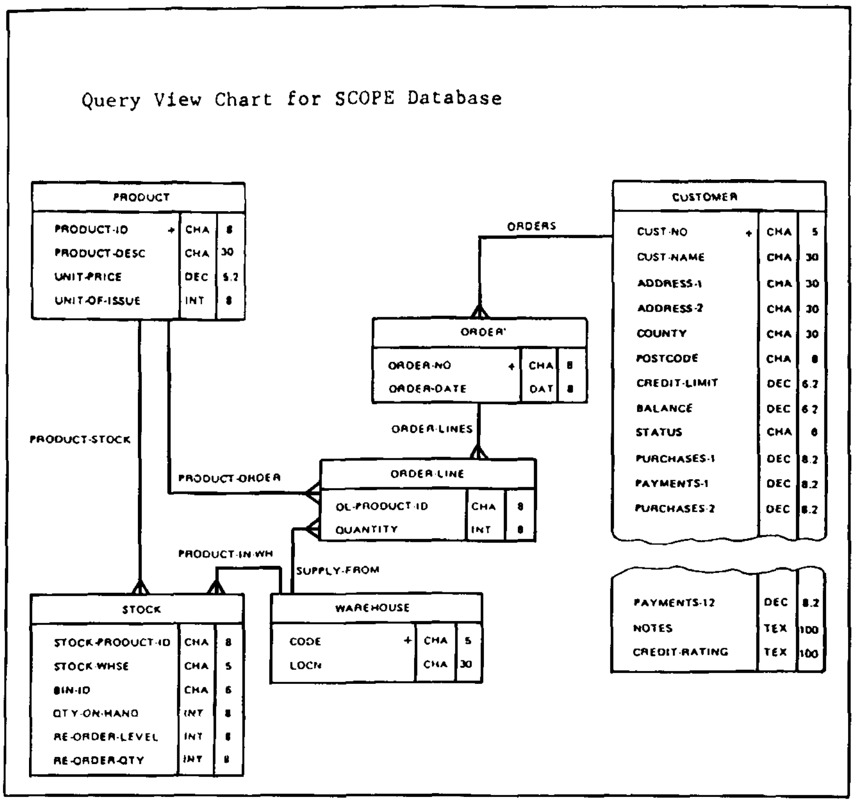


Fig. 1

3.1 Information for focus selection

For a query to be conceptually coherent means that all relations involved in that query must be linked. At a database level, these links can be pointers, or value based relationships. In this sense, a conceptually coherent query relates to a consistent subset of the database. To give an example for the SCOPE database: a query involving the relations ORDERLINE and CUSTOMER is conceptually coherent only in the case where it also refers to the relation ORDER, otherwise it is impossible to establish a link between ORDER-LINE and CUSTOMER. Intuitively this can be seen as a way of defining a notion of 'paraphrasable query' over a particular database. Note, however, that this situation is not a consequence of what the formal language will allow, since there are well-formed formal expressions which are not conceptually coherent in the sense described above.

A second intuition, based on the first one, dictates that, if a paraphrase query is expressed over a consistent subset of database relations linked by relationships, then the paraphraser can rely on this 'network' to guide the building of a coherent conceptual structure underlying the output text. This raises a question about selecting a starting point for the paraphraser within that 'network'. That starting point is the FOCUS of the query.

These intuitions can be rephrased as three assumptions:

1 Every query has a focus.
2 A focus is a relation in the current database.
3 Every relation other than the focus, involved in a particular query, must be linked directly or indirectly to that focus. This means that there is a path from the focus to every other relation mentioned in the query and such that none of the links (database relationships) making up that path refers to a relation which is not specified explicitly in the query.

The third assumption leads us to formulate a paraphrasing strategy which starts building a description of the derived relation by paraphrasing the focus. Subsequently, it paraphrases all paths from that focus stepping through each of the links that makes up such a path. The lexical material used for the description of each step is given by the model which associates, with each database relationship, an English predicate.

For example, the database relationship between ORDER and CUSTOMER can be associated with the English predicate 'to place' in the following way:

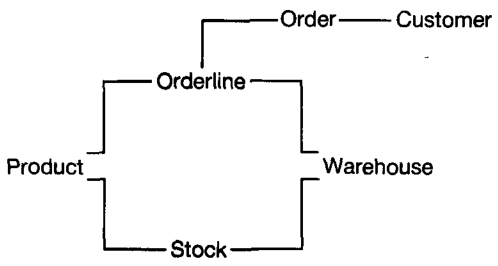
```
ORDER  ----------------------------------> CUSTOMER
                       |
                       |
                       |
                       |

          [to place; [arg 1, CUSTOMER]

                     [arg 2, ORDER    ] ].
```

where the specification of the arguments indicates that the customers place the orders.

A problem arises, however, at this point regarding directionality. Database relationships do not carry directionality and can be traversed in two directions. Given that a query is expressed over a part of the database that can be seen as a network of relations and relationships, then if the paraphraser traverses that network, circularity can occur. One must therefore impose a direction on the paths linking all relations, involved in the query, to the focus. This effectively changes the network into a tree whose root node is the focus relation. The direction in which a particular link in the tree will be traversed will then entail a different paraphrase – in some cases simply by passivising an associated English predicate (e.g. customers place orders, orders are placed by customers) or alternatively because each direction of the link is associated with a different predicate by the model.

The process for selecting a focus, therefore, can be stated as follows: If a query involves only one relation, then that relation is the focus. If it involves more than one relation, then the focus is that relation which forms the root node of a tree whose other nodes are exactly those remaining relations involved in that query.

Two practical points must be made here. The model adopted by this system stands in an elementary form. Only one flow of directionality has been imposed, with two exceptions, as illustrated below.



As a consequence, a query involving only order and customer will always have the order relation as focus. Furthermore, we predict that if two way directionality is imposed on the database relationships then, in order to avoid circularity, the two flows of direction must be kept separate.

3.2 Information for describing other database objects

In addition to allowing for the selection of an appropriate focus and for the description of directed database relationships, the model also provides for the description of other database objects. Both relations and attributes are associated with alternative descriptions (usually English nouns or complex nouns). The paraphraser will pick one of the alternatives thus specified depending on what focus has been selected for the query that is being paraphrased. For instance, the attribute CUSTOMER.CUST-NAME will be described as 'name' in a situation where the current focus is 'customer' and as 'customer name' when another focus has been picked. Similarly '<' will be translated differently depending on the attribute to which it is being applied, for example 'cheaper than' when applied to price, 'smaller than' when applied to numbers – and so on.

The conceptual type of the attributes is derived from the NEL 'End User View' model.

4 Paraphraser overview

Given an RC expression, the paraphraser will perform its task in four steps. First of all, the RC expression will be parsed into a structure which makes the syntactic build-up of the formula explicit. After completion of a basic parse, the resulting structure is converted into a list and some of its componenets are flattened out and simplified so that they can be handled more easily by the rest of the program. This is implemented as a Prolog DCG on the Essex Dec-10 using a context free grammar to specify the calculus syntax and applying its rules top-down depth first.

Secondly, the focus of the input query is determined on the basis of the relations, to which it refers, according to the stages described in the previous section. The model for the database plays, as expected, an important role in this part of the process.

In a third step, each part of the RC expression is paraphrased relative to the focus discovered previously. This part of the paraphraser, to be described in more detail in the next section, produces a conceptual/linguistic predicate argument structure which underlies the final paraphrasing text. During the fourth step the predicate/argument structure is assembled into an English text which retains some degree of explicit structure used to determine the format of what will appear on the screen. This small degree of structuring allows for the output to reflect the scope of logical operators. The original aim was to develop this component as a full LFG generator. However, although the predicate/argument structure which is input to this module of the system is compatible with LFG (as extended by Halvorsen), it was felt that the work should, in the short term, concentrate on the third stage described above, since completion of the latter was judged more critical for the success of the project as a whole. For these reasons only a basic linguistic component has been implemented which concentrates largely on agreement and word order. However, the lack of a fully implemented theoretically sound linguistic component, seems not to have impaired the quality of the paraphrases delivered. This suggests to us that, for synthesising human language text from formal languages, the implementation of a sophisticated syntactic component is subsidiary to the development of a mechanism that settles the conceptual structure underlying the final text.

5 Paraphraser strategy

The main body of the paraphraser utilises three categories of information in order to guide its actions. First of all, it analyses the syntactic structure of the incoming formal expression. Secondly, information is provided regarding the focus relation of that expression. Thirdly, both the previous items of information are used to specify the tree of database relationships and relations defined by the query.

The syntax of the relational calculus is used to determine the overall format of the paraphrasing text. A number of options are open to the user, e.g. user defined functions, ordering requirements on the retrieved information, etc. These options can be determined on the basis of the structure of the query. They are paraphrased as separate sentences which either precede or follow the text describing the main body of the formal query. A well-formed query must have a left hand side, specifying the information to be retrieved, and a right hand side constraining that information. For instance, in

```
{ (Customer.Cust-name) :( (Customer.Cust-address = 'London') ) }
                /                         \
               /                           \
              /                             \
         LHS                                 RHS
```

the left hand side specifies that customer names must be retrieved. The right hand side restricts that retrieval to the names of those customers who live in London.

This basic syntactic structure is reflected in the format of a standard paraphrase which is the following:

> FOR <DERIVED RELATION> <ACTION VERB> <TARGET LIST>

<DERIVED RELATION> is the paraphrase of the right hand side of the input query, and <TARGET LIST> of the left hand side. <ACTION VERB> is some English predicate selected on the basis of what items are contained in the left hand side (e.g. 'show' for attributes, 'calculate' for functions, etc.).

The focus of the query settles the starting point for the description of <DERIVED RELATION>. It also plays an important part in the selection of lexical material for describing database objects.

The tree of database relationships and relations which the model has assigned to the query is used, together with the syntactic structure of the query's right hand side, in structuring the description of <DERIVED RELATION>.

Overall, the paraphraser distinguishes between different kinds of comparisons that can occur on the right hand side of the formal expression. These include:

- ORDINARY comparisons, comparing the value of a database attribute with a constant.
- LINKING comparisons, comparing by means of '=' key attributes of relations between which a relationship exists in the database. These correspond to 'links' along paths in the conceptual tree as defined by the model.
- COMPLEX comparisons involving attributes of different relations without being linking comparisons.
- DISCONTINUOUS comparisons which are groups of comparisons bundled together under a different logical operator from that of the previous level.

The paraphraser starts by describing the focus of the query. This involves not only a paraphrase of the focus relation itself, but also of ordinary comparisons involving an attribute of that relation. In the next step, all linking comparisons between the focus and other relations one step along the paths in the conceptual tree are paraphrased. When one such link is described, the relation newly linked to the old focus is propagated as a subsidiary focus. This new focus is described in turn, including links to other relations further along the path, which will in time also become subsidiary foci. When all the links along a path have been described, the old focus is (recursively) restored. All partial paraphrases are linked together by means of the appropriate logical operators. Paraphrasing is thus done recursively, relative to the syntactic structure of formal expression components, the focus of the query and the conceptual tree delivered by the model.

For the top level focus, all four types of comparisons are described in turn. However, for subsidiary foci, complex comparisons are omitted. They typically involve two relations and it is difficult to decide at which stage they should be paraphrased. All complex comparisons are therefore paraphrased relative to the overall, top level focus.

The elements of the left hand side are described by retrieving from the model the appropriate lexical material with which they are associated relative to the overall focus of the query. The descriptions of similar objects are conjoined and grouped with an appropriate verb. Such verb phrases, if applicable, can also be conjoined.

6 An example

To illustrate the operation of REMIT we give a comprehensive query example, defined on the SCOPE database, showing each stage of the transduction and paraphrasing process. This is one of many such examples compiled jointly by ICL and Essex to test the different features of the software.

Querymaster:

> List stock.stock-whse,bin-id,stock-value is qty-on-hand*unit-price sorted by ascending warehouse.locn where warehouse.locn > 'London' and re-order-qty < 100 and product-stock and product-in-wh starting stock

Relational calculus:

> {stock-value(stock.qty-on-hand,product.unit-price)
> := ('stock.quantity-on-hand*product.unit-price')
> w(stock.stock-whse,stock.bin-id,stock-value(stock.qty-on-hand,
> product.unit-price)):

```
(( wh  warehouse)(((wh.locn>'London')(stock.reorder-qty<100))
   ((stock.whse = wh.code)(stock.product-id = product.product-id))))
up(wh.locn)}
```

Computed focus:

= Product (Note that Product is not referred to in the Target List)

Paraphrase:

```
For products
   which are physically stocked
      and whose re-order-qty is less than 100
   and which are stored in warehouses
      whose location is alphabetically listed after 'London'
(1) show
      (a) the warehouse codes
      (b) the bin numbers

and

(2) calculate and display stock-value

where stock-value is defined as (qty-on-hand*unit-price).

Sequence the result by ascending warehouse location.
```

7 Conclusion

This paper has described a prototype paraphraser developed as an ICL project at the University of Essex and fully implemented in Prolog on a DEC10. The paraphrasing process has been split into two steps using an extension of the relational calculus as an intermediary representation; this is in order to enhance the portability of the paraphraser over relationally complete query languages. The system has been successfully tested for a wide range of sample queries and results have both justified the extensive efforts spent in defining a suitable model and also underlined the importance of selecting an appropriate focus to guide the paraphrasing process. Furthermore it has been shown that the provision of a sophisticated NL grammar formalism is subsidiary to the development of a mechanism for defining the underlying, coherent and unambiguous conceptual structure of the output paraphrase. Overall the system has demonstrated that it is feasible to deliver paraphrases of formal query language expressions which are helpful to the user in verifying his natural language intention.

Acknowledgment

The authors are indebted to V. West of International Computers Limited for many helpful discussions on this work. Funding for this research has been provided by International Computers Limited.

References

1 CODD, E.F.: A Database Sublanguage founded on the Relational Calculus Proceedings of the ACM SIGFIDET Workshop on Data Description, Access and Control, 1971.

2 CODD, E.F.: Relational Completeness of Database Sublanguages in Database Systems. Courant Computer Science Series, Vol. 6, Prentice Hall, Englewood Cliffs, 1972.

3 DATE, C.J.: An Introduction to Database Systems, Addison-Wesley, 1977.

4 HALVORSEN, P.K.: Semantics for Lexical Functional Grammar in Linguistic Inquiry, Vol. 14, No. 4, pp. 567–615, 1983.

5 International Computers Ltd., Using Querymaster (200 level), Publication R00260/00, 1983.

6 International Computers Ltd., Using Querymaster (QM.250) Publication R00433/01, 1985.

7 KAPLAN, R. and BRESNAN, J.: Lexical Functional Grammar. A Formal System for Grammatical Representation in J. Bresnan (Ed.), The Mental Representation of Grammatical Relations, MIT Press, Cambridge (Mass.), 1982.

8 McKEOWN, K.: Paraphrasing using given and new Information in a Question Answering System in Proceedings of the 17th ACL, La Jolla, 1979.

9 SHEPHARD, I.: Implementation of a Transduction Algorithm to Convert Querymaster Language Statements into Relational Calculus, M.Sc. Dissertation, University of Essex, 1985.

10 THOMAS, J.C. and GOULD, J.D.: A Psychological Study of Query by Example, Proceedings NCC 44, 1975.

11 ULLMAN, J.D.: Principles of Database Systems, Computer Science Press, 1980.

12 WALLACE, M. and WEST, V.: QPROC: A Natural Language Database Enquiry System Implemented in PROLOG, ICL Technical Journal, November 1983.

13 WEST, V.: Natural language database enquiry. ICL Technical Journal, May 1986.

Natural language database enquiry

V. West
ICL Applied Systems, 33 Kings Road, Reading, Berkshire

Abstract

This paper describes the results of a research project to produce a natural enquiry language system (NEL) by adding a PROLOG front-end to Querymaster, the standard ICL program for querying information stored in a database. The system architecture and the role of the 'knowledge engineer' in preparing the database are discussed. A sample session is included, user experience is reviewed and some possible extensions are discussed.

1 Introduction

An earlier paper[1] in this journal described QPROC – an interactive natural language enquiry system providing access to a relational database. The main aim of that paper was to demonstrate the practical use of the PROLOG implementation language[2] on an application of some complexity. A major limitation of QPROC was that the database itself had to be implemented in PROLOG, and so only small 'toy' applications could be implemented. A system capable of accessing 'real' databases would be of much greater interest.

The standard ICL program for querying information stored in a database is Querymaster[3]. This provides access to various types of database including IDMS or COBOL files, and CAFS (Content Addressable File Store) high-speed facilities can be used. The user expresses his questions in an easy to use computer language.

This paper describes the results of a research project to produce a natural enquiry language system (NEL) by adding a PROLOG front-end to Querymaster, running under VME on the ICL 2900. ICL-PROLOG[4] was used and NEL was able to profit from the QPROC experience.

The paper covers:

- the system architecture (section 2)
- the role of the 'knowledge engineer' who prepares a database for natural language enquiry, and the NEL data model (section 3)
- a commentary (section 4) on the sample NEL session in Appendix B. This

uses an IDMS database SCOPE (Stock, Customer, Order and Product Enquiries) which represents an ordering system for a stationery business. This database and its associated natural language vocabulary is described in more detail in Appendix A, and is used to provide examples throughout this paper.
- the result of practical experience with NEL (section 6)
- some possible extensions (section 7).

Another paper[5] in the same issue of this journal describes related work at the University of Essex supported by an ICL University Research Council (URC) grant. That work is concerned with the generation of natural language paraphrases of formal database queries.

2 Architecture

Fig. 1 illustrates the run-time architecture.

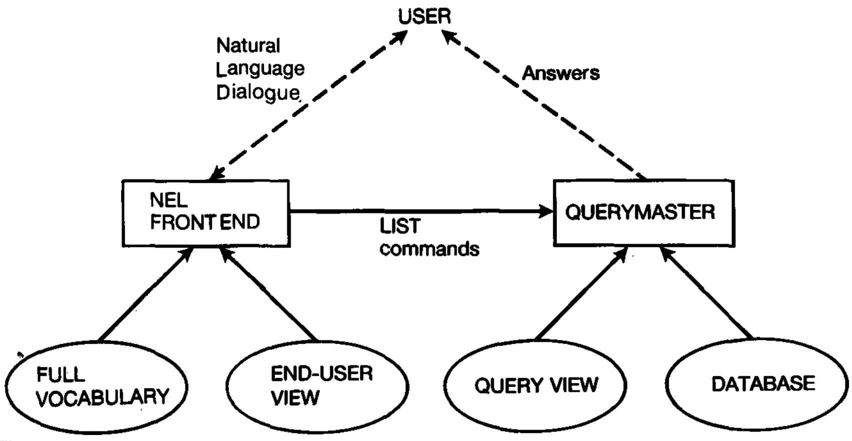


Fig. 1

The NEL front-end receives natural language questions from the user. Using the Full Vocabulary and End-User View for the subject prepared by the 'knowledge engineer' it attempts to understand the question and generate an equivalent Querymaster (LIST) command. Some questions such as 'help' it answers without recourse to Querymaster. The Full Vocabulary contains all the words and phrases for the subject and the End-User View defines the subject using the NEL Data Model. Both are described in section 3, which explains the role of the knowledge engineer.

LIST commands are passed across to Querymaster which accesses the database and returns answers directly to the user. Querymaster uses the 'Query View' which provides a description of the database.

The front-end consists of eleven components, 7 'natural language independent' and 4 'natural language dependent' ones. The former will be the same for different languages, whereas the latter will have different versions for different languages. The separation into components aims to maximise the independent ones (from now on the paper relates to English only).

The components are (natural language dependent ones being distinguished by an asterisk):

SESSION CONTROL	– contains the entry point to NEL and is responsible for the session dialogue with the user and for calling the BACK-END component to start and end Querymaster sessions. The 'sessions' may potentially be with different databases but NEL does not support a change of database.
QUERY CONTROL	– controls the processing of all the user's queries in a session.
SCANNER*	– is responsible for all input. It splits natural language questions into words.
RESPONDER*	– is used for outputting all messages to the user.
PARSER CONTROL	– looks up the meaning of the words in a question using the WORD PATTERNS component and the Full Vocabulary. It calls the PARSER component to interpret the question and classify it as a database query or a 'meta query' (about the description of the data rather than the data itself).
WORD PATTERNS*	– recognises word endings (including plurals) and patterns (numbers, dates, times and money).
PARSER*	– attempts to interpret the question into a formal representation of its meaning. This representation called 'descriptions and qualifiers' is a development of that used in QPROC[1,6] and will not be described further in the present paper. The Parser is implemented in PROLOG grammar rule notation[2] and provides a wide coverage of English grammar. Noun phrases are parsed 'breadth-first': all alternative interpretations are found, not just the first, so that next time the Parser needs to find a noun phrase starting at the same point in the question, it need not parse it again.
META-QUERY HANDLER	– is responsible for answering meta-queries, though currently the only meta-query supported is 'help'.
DATABASE QUERY HANDLER	– is responsible for answering database queries.
CONVERTER	– converts the 'descriptions and qualifiers' for a database query into a Querymaster LIST command.
BACK-END	– handles the interface with Querymaster. Querymaster is implemented mainly in Pascal.

3 The role of the knowledge engineer

For any database to be accessed using NEL, a 'knowledge engineer' must carry out some preparatory work; it is essential that this knowledge engineer understands the structure and content of the database, and the related natural language vocabulary used by its intended users.

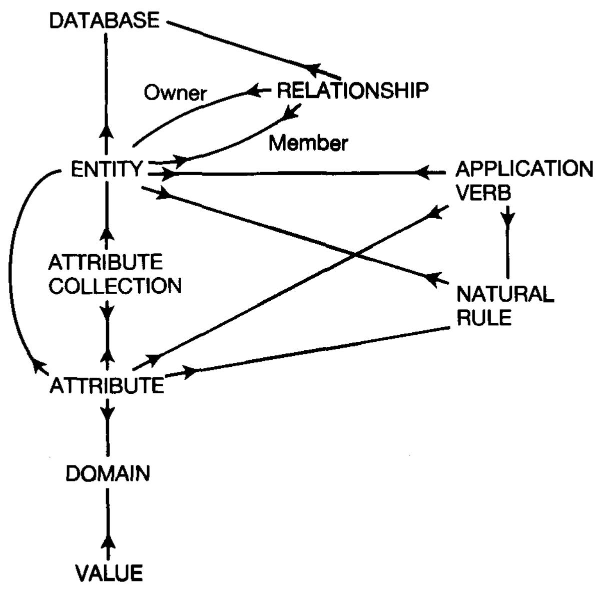


Fig. 2

The knowledge engineer prepares the End-User View which defines the subject (database) using the NEL Data Model, and the Full Vocabulary which contains all the words and phrases for the subject. The process will typically be an iterative one in response to user requests for changes including additional vocabulary.

3.1 The End-User View

Fig. 2 illustrates the object-types of the NEL Data Model and the relationships between them (the 'crow's foot' sign ⋏ indicates a 'many' relationship). It is based on the Entity-Attribute-Relationship model with the following additional object-types:

Attribute Collection
Application Verb
Natural Rule

An *attribute collection* consists of two or more attributes from the same entity in the order in which they appear together in natural language phrases. For example, the attribute collection 'address' in SCOPE consists of the customer attributes address-1, address-2, county and postcode.

An *application verb* defines the use of the verb in natural language questions. For example:

a customer *places* an order for a product

The definition includes a 'case' list and a reference to a natural rule.

Each entry in the case list describes a case and its corresponding entity or attribute. The case list for 'place' is:

subject	:	entity customer
first object	:	entity order
for-object	:	entity product

(for verbs with more than one object like 'give' a second object can be defined, and various prepositional objects are provided). This description assumes the verb is in the active mood. NEL automatically performs any permutation required when the verb is used in the passive.

A *natural rule* has several uses. For an application verb it can provide the associated Querymaster access path. For 'place' this is defined by the relationships:

orders and order-lines and product-order

In addition to these new object-types, additional information is required for others:

For *database*, a natural language *explanation* of the database for use in 'help' messages. Such explanations could usefully be added to other object-types.

For *entity* and *attribute collection*, a *category* (see below).

For *attribute*, a *response-default* indicator which determines which attributes are to be included in the answers when the question does not specifically identify them. For example the response-defaults for 'customer' are the number and name. *Result attributes* to be calculated may also be defined by the knowledge engineer. So for example 'unused credit' is defined as:

(customer) credit-limit – balance

For *domain*, a *category* which matches various common natural language words and phrases. The categories are:

person	– matching 'who' etc.
thing	– 'what', 'which'
organisation	= person or thing
count	– (none)
measure	– 'how ---'
money	– 'how much'
location	– 'where'
timepoint	– 'when'
reason	– 'why'
method	– 'how'

For categories 'measure' and 'money', units may be provided: for money the unit is 'pounds'.

Categories may also be provided for entities and attribute collections.

Values may be provided for character type domains. Some or all of the possible values may be given, and may then be used as words in natural language questions.

3.2 The Full Vocabulary

Fig. 3 illustrates the generation of the Full Vocabulary. The knowledge engineer uses a program called Vocabulary Builder which requires:

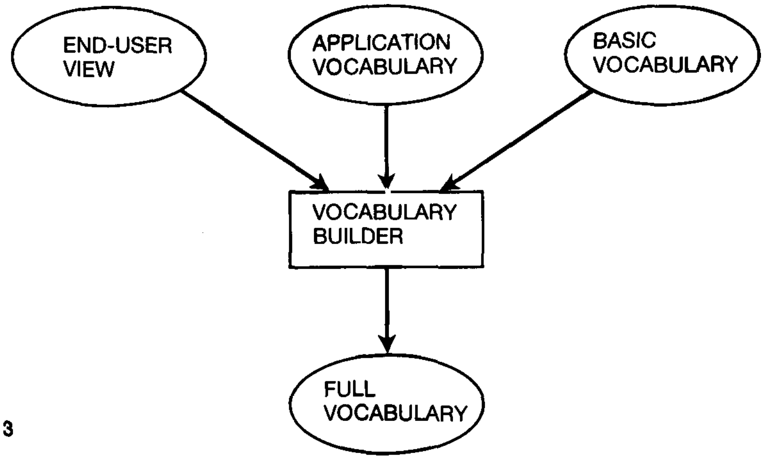


Fig. 3

the End-User View for the subject (see above)
the Application Vocabulary for the subject
the Basic Vocabulary (the same for every subject)

The Basic Vocabulary is included with NEL and defines all the subject independent words. It contains approximately 130 common words, phrases and abbreviations including for example:

a
and
bigger than
do
equal to OR eq OR =
for
get
is
our
someone
who

The Full Vocabulary will contain the following words from the End-User View:

nouns–database, entities, attribute collections, attributes, units and values
verbs– application verbs

For nouns NEL recognises plurals formed in the standard way using -s -es or -ies; for a compound noun such as 'unit price' the plural would be recognised as 'unit prices'. For verbs it recognises the standard endings -s -es -ies -ing -ed -ied -en, allowing for doubling of final consonants.

The Application Vocabulary is where the knowledge engineer provides additional synonyms/abbreviations or irregular inflexions. For nouns he may define irregular plural forms and for verbs irregular tenses. For example 'client' is a synonym of 'customer', 'units of issue' is the plural of 'unit of issue' and 'held' is the past of 'hold'.

4 Commentary on a sample session

This commentary relates to the sample session in Appendix B. The questions there have been numbered for reference in this commentary (for technical reasons the numbers start at 6)

(6–9) After the sign-on sequence, NEL outputs the explanation of the subject and the entities it covers.

(10) The response 'OK' shows that NEL has understood the question. The generated Querymaster command is also displayed (this is optional). Notice that only the customer numbers and names are output as they are the response-default attributes. Here and later the output has been abbreviated (indicated by ...) for brevity.

(11) 'address' is an attribute collection. The output is in 'labelled' format rather than 'tabular' as it would be too wide for the latter.

(12) 'unused credit' is a result attribute.

(13–14) 'Berks' was provided as a 'county' value, but 'Devon' was not; so 'Devon' must appear in quotes in the exact case in which it appears in the database and be qualified by 'county'.

(15–16) 'credit limit' and 'balance' are included in the answers because they occur in comparisons (see section 5).

(18) Note the irregular past tense 'held'.

(20) A better answer would be simply 'NO' (see section 5).

(23) 'December figures' is an attribute collection. 'LMBC' is a synonym for a customer name value.

(24) NEL ignores words it does not understand.

(25) NEL cannot understand this – products do not have addresses – so it outputs an error message instead of 'OK'.

(26) NEL can understand the question and so responds 'OK' but cannot answer it as Querymaster will not accept such questions.

(27) The 'help' output is similar to the sign-on output (see above), 'recall' can also be used to display the previous question for possible editing and resubmission.

5 Practical experience

Users have reacted very favourably. It gives them a lot of freedom in expressing questions. Natural language is more familiar than a computer language, and to the surprise of many users has proved to be more concise – typically natural language questions are half the length of Querymaster ones.

The extra facilities deriving from the extended data model described in section 3 were well received, especially attribute collections, response-defaults, result attributes, verbs and synonyms. These allow the user to express his question concisely and to control the answers he receives. Many of these extensions could also be profitably included in formal query languages.

The heuristics needed to decide what should be included in a question are not easy to construct. We all know people who provide too little or too much in answer to questions. NEL currently includes in its answers:

- anything specifically requested by 'list' (or a synonym):
  List the *customers*
  Customer names (here 'list' is understood)
- anything distinguished by a questioning word like 'who' or 'which':
  Which *warehouses* hold ash trays?
- any compared attribute:
  ... *credit limits* over £50
- any attribute for which alternative values are given:
  ... Berks and Hants customers (*county* is output)

If none of these apply the first entity encountered is output:

Are there any Essex *customers*?

Also, unless specific attributes are mentioned, entity output is restricted to the response-default attributes.

As noted in section 2, answers are returned directly to the user by Querymaster. On the whole this works well, as can be seen from Appendix B. However headings are not always necessary, and some questions such as:

Are ash trays 83p?

could be better answered by a simple 'yes/no', and others like:

Do any products have a price over £1?

by a 'yes/no' plus, for yes, a set of answers.

The limitation of users to a small Basic Vocabulary did not cause any difficulties, but the ease with which new synonyms could be added was very significant as a wider vocabulary made the system appear much better to users.

The coverage of English grammar is wide, as can be seen from Appendix B, but it is not exhaustive, and this creates a difficulty. When NEL does not understand a question the user may well type in a more complicated one to 'clarify' his intention and this also is not understood. One way around this difficulty would be for the user documentation to describe the grammar supported, but this is inappropriate as such a description would be complex and against the spirit of a natural language interface. A possible solution is to list those constructions the system does *not* accept which the user might try e.g. 'conjoined' verbs as in:

Which warehouse holds a calculator and supplied a rack?

6 Possible extensions

Currently *ambiguous* questions are not detected as such – the first interpretation found is the one taken. They should be referred back to the user for clarification. This requires that the different meanings can be presented for him to choose which he meant. Even where there is no ambiguity it is useful for the user to be able to check whether his question has been interpreted in the way he intended. Currently the only 'meaning' which can be presented to the user is the generated Querymaster command, but a natural language paraphrase is much to be preferred^5.

No use is made of *context* to support "conversation": questions are treated separately. The most important contextual features are the use of pronouns as in:

Who are our Berks customers?
What are *their* credit limits?

and ellipsis:

Who placed orders on 7.4.80?
9.4.80?

Further *meta-question* facilities would be useful. These might include questions about:

- the attributes for an entity
- the values for an attribute
- the verbs for an entity
- the words in the vocabulary

Meta-questions can be recognised in various ways. Compare:

What is a product?
Tell me the products

The former could be interpreted as a meta-question, a request for general information about products, and the latter as a data question, one for specific information about particular products. However the simplest solution is to use a keyword such as 'help' to distinguish meta-questions.

Instead of ignoring unrecognised words, the user could be asked to correct them. Spelling correction is a further refinement. Users would also like to be able to extend the vocabulary by statements like:

line means product

This raises issues about the permanence and scope of the extension (is it available at later sessions, and to other users?)

Some Querymaster features are not available through natural language, though there is an escape facility allowing direct input of any Querymaster command. These include functions like 'maximum', ordering of answers and explicit arithmetic, all of which could be added to natural language, e.g.:

What is the highest product price?
Show the customers in alphabetical order
List price/unit of issue

7 Conclusion

The ideas first developed in QPROC[1] which provided natural language enquiry capabilities on 'toy' databases, have been successfully extended in NEL which provides enquiries on real databases. NEL offers a very acceptable natural language interface to its users.

Acknowledgements

Among those who have contributed to the NEL project, I would particularly like to thank Mark Wallace (now at the European Computer-Industry Research Centre, Munich) who provided much of the design and implementation; and Philip Pearson who also assisted in the implementation.

References

1 WALLACE, M.G. and WEST, V.: 'QPROC: a natural language database enquiry system implemented in PROLOG', ICL Tech. J., 3, (4), 393–406.
2 CLOCKSIN, W.F. and MELLISH, C.S.: 'Programming in PROLOG', 2nd edition, Springer-Verlag, 1984.
3 International Computers Limited: 'Using Querymaster (QM.250)', Technical Publication R00433/01, ICL House, Putney, London 1985.
4 International Computers Limited: 'ICL PROLOG User Guide', Technical Publication R30036/01, ICL House, Putney, London 1985.
5 (Reference to University of Essex paper in same issue of ICL Technical Journal).
6 WALLACE, M.G.: 'Communicating with Databases in Natural Language', Ellis Horwood Limited, 1984.

Appendix A – The SCOPE database

SCOPE represents an ordering system for a stationery business and covers:

products sold by the company
customers
orders placed by customers
orderlines which make up orders
warehouses used by the company
stocks held at warehouses

The SCOPE vocabulary in addition to the basic vocabulary (OR indicates synonyms) is:

Entities:

product OR item
order-line OR order line OR orderline
order
customer OR client
warehouse
stock

Attributes for each entity (* response default, †result attribute)

Product

*product-id OR code
*product-desc OR description
– specified descriptions: ash tray
blotter
calculator
pencil sharpener OR sharpener
printout filing rack OR rack
staple extractor
unit-price OR unit price OR cost OR price (in pounds)
unit-of-issue OR unit of issue

Order-line

*ol-product-id OR code
*quantity
†value (in pounds)

Order

*order-no OR number
*order-date OR date

Customer

*cust-no OR number
*cust-name OR name
– specified names: Green Vegetable Associates OR GVA
Long Mile Bus Company OR LMBC
Virtual Machines Limited OR VML

address-1
address-2 OR town
county
– specified counties: Berks OR Berkshire
Essex
Hamps OR Hants OR Hampshire address
Herts OR Hertfordshire
Lancs OR Lancashire
Leics OR Leicestershire
Notts OR Nottinghamshire

postcode
credit-limit OR credit limit OR credit ceiling (in pounds)
balance (in pounds)
status

- specified statuses: open, closed

year-id-1
month-id-1
purchases-1 }
payments-1 } oct-figures OR October figures (in pounds)
...
year-id-12
month-id-12
purchases-12 }
payments-12 } sep-figures OR September figures (in pounds)
text-length
notes
credit-rating OR credit rating
†unused-credit OR unused credit (in pounds)

Warehouse

*code
- specified codes: BRA01 OR Bracknell
  USA26 OR USA
  IRQ01 OR Iraq

*locn OR address OR location

Stock

*stock-product-id OR code
*stock-whse
- specified warehouse codes: see above

*bin-id OR bin

qty-on-hand OR quantity on hand }
re-order-level OR re-order level OR reorder level }
buffer-stk-level OR buffer stock level }
re-order-qty OR re-order quantity OR reorder quantity } re-order-details OR re-order details OR reorder details

†stock-value OR value (in pounds)

Verbs

an order *contains* a product
a warehouse *holds* stock
a warehouse *holds* a product
a customer *orders* a product
a customer *places* an order for a product
a warehouse *supplies* a product
an orderline *is worth* a sum of money

The relationships, which are not part of the vocabulary are:

relationship	*owner*	*member*
PRODUCT-ORDER	PRODUCT	ORDER-LINE
PRODUCT-STOCK	PRODUCT	STOCK
ORDER-LINES	ORDER	ORDER-LINE
ORDERS	CUSTOMER	ORDER
PRODUCT-IN-WH	WAREHOUSE	STOCK
SUPPLY-FROM	WAREHOUSE	ORDER-LINE)

Appendix B

```
GOOD AFTERNOON.
IT IS 14: 3: 18 ON 6. 1. 86.

        THIS IS THE NATURAL ENQUIRY LANGUAGE SYSTEM VERSION 20.30/SCOPE2.1.
                              (C) COPYRIGHT ICL 1985.

WHICH SUBJECT WOULD YOU LIKE TO ASK QUESTIONS ABOUT?
PLEASE ENTER THE SUBJECT NAME OR 'bye'.
0006 scope2

PLEASE ENTER A NAME FOR THIS SESSION.
0007 vw

PLEASE ENTER YOUR USER NAME.
0008 rayjones

PLEASE ENTER THE PASSWORD.
(IF THERE IS NO PASSWORD ENTER 'none'.)
0009 secret

PLEASE WAIT A MOMENT.

SCOPE2 STANDS FOR STOCK, CUSTOMER, ORDER AND PRODUCT ENQUIRIES.
SCOPE2 COVERS:
              PRODUCTS
              ORDER-LINES
              ORDERS
              CUSTOMERS
              WAREHOUSES
              STOCKS

YOU MAY NOW ASK QUESTIONS ABOUT SCOPE2.
WHEN YOU HAVE FINISHED ENTER 'end'.
0010 Who are the customers?
OK.
LIST CUSTOMER.CUST-NO,CUST-NAME

PLEASE WAIT A MOMENT.
 CUSTOMER
   CUST-NO   CUST-NAME
-----------------------------------------------
   02109     SPRING SURPRISE CHOCOLATES
   02613     NON STICK TYRE COMPANY
   04377     LONG EARED FROG COMPANY
   ...

PLEASE ENTER YOUR NEXT QUESTION.
0011 List the customer names and addresses.
OK.
LIST CUSTOMER.CUST-NAME,CUSTOMER.ADDRESS-1,ADDRESS-2,COUNTY,POSTCODE
```

```
  CUSTOMER
    CUST-NAME: SPRING SURPRISE CHOCOLATES
    ADDRESS-1: 1 FLAXMAN ROAD
    ADDRESS-2: BASSETLAW
  COUNTY: NOTTS                                                              POSTCODE: N45 91A
  ...

PLEASE ENTER YOUR NEXT QUESTION.
0012 What are the customer names and their unused credits?
OK.
LIST CUSTOMER.CUST-NAME,UNUSED-CREDIT IS CUSTOMER.CREDIT-LIMIT – BALANCE

  CUST-NAME                              UNUSED-CREDIT
--------------------------------------------------------
  SPRING SURPRISE CHOCALATES                    100.00
  NON STICK TYRE COMPANY                         78.50
  LONG EARED FROG COMPANY                       100.00
  ...
PLEASE ENTER YOUR NEXT QUESTION.
0013 The Berks customers.
OK.
LIST CUSTOMER.CUST-NO,CUST-NAME WHERE CUSTOMER.COUNTY = 'BERKS'

  CUSTOMER
    CUST-NO    CUST-NAME
--------------------------------------------------------
    02802      55TH FLOOR HOUSING CO LTD.
    02865      OXTON WATER SUPPLY ASSOC.
    04692      STAPLES COMPLAINT LTD.
    10525      EVEREST EXPLORERS UNLIMITED
    99999      NATIONAL DRINKERS (BERKS)

PLEASE ENTER YOUR NEXT QUESTION.
0014 Tell me the customers for the county of 'DEVON'.
OK.
LIST CUSTOMER.CUST-NO,CUST-NAME WHERE CUSTOMER.COUNTY = 'DEVON'

  CUSTOMER
    CUST-NO    CUST-NAME
--------------------------------------------------------
    07521      GRADWELL SHOE CORPORATION

PLEASE ENTER YOUR NEXT QUESTION.
0015 Which customers have credit limits over £50 and under £150?
OK.

LIST   CUSTOMER.CREDIT-LIMIT,CUSTOMER.CUST-NO,CUST-NAME   WHERE   CUSTOMER.CREDIT-
LIMIT > 50 AND CUSTOMER.CREDIT-LIMIT < 150

  CUSTOMER
    CREDIT-LIMIT    CUST-NO    CUST-NAME
----------------------------------------------------------------------
    100.00          02109      SPRING SURPRISE CHOCOLATES
    100.00          02613      NON STICK TYRE COMPANY
    100.00          04377      LONG EARED FROG COMPANY

    ...
PLEASE ENTER YOUR NEXT QUESTION.
0016 Which customers have balances over their credit limits?
OK.

LIST   CUSTOMER.BALANCE,CUSTOMER.CUST.NO,CUST-NAME   WHERE   CUSTOMER.BALANCE >
CUSTOMER.CREDIT-LIMIT
```

```
    CUSTOMER   CUST-NO     CUST-NAME
    BALANCE
-------------------------------------------------------------
    1035.10    07143       ZERO-C SCREEN CLEANERS
     149.60    03873       BLUE & GREY BOXES CORPORATION
     244.40    06513       INDEXED SEQUENTIAL STOCKS LTD
     151.90    05763       SAFE STRING MANUFACTURERS

PLEASE ENTER YOUR NEXT QUESTION.
0017 Who placed what orders on 07.04.80?
OK.

LIST   ORDER,CUSTOMER.CUST-NO,CUST-NAME   WHERE   ORDER.ORDER-DATE = 07.04.80   AND
(ORDERS AND ORDER-LINES AND PRODUCT-ORDER)

    ORDER-NO     ORDER-DATE    CUST-NO    CUST-NAME
------------------------------------------------------------------------
    D 000102     07.04.80      01857      WINE GROWERS GROUP INCORP.
    D 000109     07.04.80      03180      LINE PAPER PRODUCTS AND CO.
    D 000108     07.04.80      03117      FOREIGN EXCHANGE COMPANY LTD
    ...

PLEASE ENTER YOUR NEXT QUESTION.
0018 Where are ash trays held?
OK.

LIST WAREHOUSE WHERE (PRODUCT-IN-WH AND PRODUCT-STOCK) AND PRODUCT.PRODUCT-DES-
C = 'ASH TRAY'

  WAREHOUSE
    CODE      LOCN
--------------------------------------------
    IRQ01     PO BOX 640 BAGHDAD IRAQ
    USA26     CLAIRE RD E.BRUNSWICK NJ USA
    BRA01     LOVELACE RD BRACKNELL BERKS

PLEASE ENTER YOUR NEXT QUESTION.
0019 What orders were placed for blotters?
OK.

LIST ORDER WHERE PRODUCT.PRODUCT-DESC = 'BLOTTER' AND (ORDERS AND ORDER-LINES AND
PRODUCT-ORDER)

  ORDER
    ORDER-NO     ORDER-DATE
-----------------------------
    A 000007     01.04.80
    A 000010     01.04.80
    A 000030     01.04.80
    ...

PLEASE ENTER YOUR NEXT QUESTION.
0020 Do ash trays have a price over £1?
OK.

LIST   PRODUCT.UNIT-PRICE   WHERE   PRODUCT.UNIT-PRICE > 1   AND   PRODUCT.PRODUCT-
DESC = 'ASH TRAY'

  PRODUCT
    UNIT-PRICE
------------------
THERE ARE NONE.
```

```
PLEASE ENTER YOUR NEXT QUESTION.
0021 What price are blotters?
OK.

LIST PRODUCT.UNIT-PRICE WHERE PRODUCT.PRODUCT-DESC = 'BLOTTER'

  PRODUCT
    UNIT-PRICE
---------------------
    9.03

PLEASE ENTER YOUR NEXT QUESTION.
0022 Are there any Essex customers?
OK.

LIST CUSTOMER.CUST-NO,CUST-NAME WHERE CUSTOMER.COUNTY = 'ESSEX'

  CUSTOMER
    CUST-NO      CUST-NAME
-------------------------------------------------------
    04944        THE POTTER FOSTER-HOMES
    03243        LONG MILE BUS COMPANY
    02424        CASH MART DISCOUNTS LTD
    04881        ZIPPETY DOODAH LTD.
    03306        THREE MILE ISLAND SHELTER CORP
    03180        LINE PAPER PRODUCTS AND CO.
    04818        SOUTHGATE SECURITIES CO LTD

PLEASE ENTER YOUR NEXT QUESTION.
0023 December figures for LMBC.
OK.

LIST CUSTOMER.PURCHASES-3,PAYMENTS-3 WHERE CUSTOMER.CUST-NAME = 'LONG MILE BUS
COMPANY'

  CUSTOMER
    PURCHASES-3      PAYMENTS-3
-------------------------------------
    14.60            14.60

PLEASE ENTER YOUR NEXT QUESTION.
0024 Good Berks customers.
I DO NOT UNDERSTAND "Good".
I SHALL TRY TO UNDERSTAND YOUR QUESTION WITHOUT IT.
OK.

LIST CUSTOMER.CUST-NO,CUST-NAME WHERE CUSTOMER.COUNTY = 'BERKS'

  CUSTOMER
    CUST-NO      CUST-NAME
---------------------------------------------------
    02802        55TH FLOOR HOUSING CO LTD.
    02865        OXTON WATER SUPPLY ASSOC.
    04692        STAPLES COMPLAINT LTD.
    10525        EVEREST EXPLORERS UNLIMITED
    99999        NATIONAL DRINKERS (BERKS)

PLEASE ENTER YOUR NEXT QUESTION.
0025 Product addresses.
I CANNOT UNDERSTAND WHAT "Product addresses" REFERS TO.

PLEASE ENTER YOUR NEXT QUESTION.
0026 List 20 customers.
OK.
I CANNOT ANSWER QUESTIONS WHICH USE COUNTS IN THIS WAY.
```

```
PLEASE ENTER YOUR NEXT QUESTION.
0027 Help!
OK.
THE CURRENT SUBJECT IS SCOPE2.
SCOPE2 STANDS FOR STOCK, CUSTOMER, ORDER AND PRODUCT ENQUIRIES.
SCOPE2 COVERS:
                    PRODUCTS
                    ORDER-LINES
                    ORDERS
                    CUSTOMERS
                    WAREHOUSES
                    STOCKS

YOU MAY ASK QUESTIONS ABOUT SCOPE2.
IF YOU HAVE FINISHED ENTER 'end'.

PLEASE ENTER YOUR NEXT QUESTION.
0028 end.
OK.

WHICH SUBJECT WOULD YOU LIKE TO ASK QUESTIONS ABOUT NOW?
PLEASE ENTER THE SUBJECT NAME OR 'bye'.
0029 bye

GOODBYE.
```

The *me too* method of software design

Peter Henderson
University of Stirling
Cydney Minkowitz
Software Engineering Technology Centre
International Computers Ltd, Kidsgrove

Abstract

The *me too* method addresses two concerns in the process of software design. The first is effective communication of a design specification amongst members of a design team. The second is early feedback on the correctness and adequacy of a design. The first concern is achieved by using mathematics as a means of precisely stating design decisions as they are made. The second is accomplished by transforming the specification into an executable design prototype to be tested.

Testing a prototype uncovers design errors and misconceptions. The faults are explicitly recorded in the formal specification and so can be readily attended to. The *me too* method encourages a team to revise the specification and to subject the new design to further inspection by execution. The method is therefore an iterative one. The emphasis on iteration enables exploration of alternative design decisions and early detection of design faults.

This paper describes the *me too* method and illustrates its use in the design of a simple prototype of a system for handling production rules.

1 Introduction

This paper addresses itself to the problem of software design. All too often when we design software we proceed to implementation stages before the design itself is complete. The reasons for this are simple. To design the software completely implies that we have a precise and undeniable statement of what that software is to do. Such a statement would allow an outsider to ask us a question of the form 'what will happen if, using your software, I were to ...' and would enable us to give an accurate response. In conventional software design we are not usually in this happy position, for until we have an implementation of our proposed software. we have probably not completely decided what it will do in all circumstances. We have no method of making a precise statement of those design decisions we do make, short of completing the implementation itself.

Here we propose that one realistic remedy to this problem is to make use of mathematics as a language for software design. We use mathematics as a programming language. Hence we design by completing a description of the software in a mathematical language. We ensure that this language is executable by machine. But we pay little attention to the performance or user-interface of this implementation, the objective being to achieve early in the design process and at very low cost, a precise description of the design which can be used to answer the questions of the outsider.

We concern ourselves particularly with the problems that arise when software is to be designed by a team. The team must reach decisions, and agree on them before moving on to implementation. Mathematics is a language which the design team can use for communication amongst themselves. They can use this language to make precise statements of proposals which can then be argued about. They can investigate alternatives with each design choice being carefully and precisely noted and considered. The necessary mathematics required to achieve this purpose is very elementary and can be learned in a few days. We give an example of its use in this paper.

Our starting point is functional programming. Programs written in a purely functional style are considered comparatively easy to understand. This is because functional programs obey the universal laws of mathematics.

Many of the principal components of a software system can be described using functional programs. In fact, many components can be described using mathematical functions. For example, a program component which is to define the displayed value of an updated screen for a given screen and a given input might be defined by a function f(screen, input) which determines every field to appear on the updated screen solely from the original value of the screen and from the value of the given input.

The virtue in describing expected computational behaviour using mathematical functions has two facets. Firstly, mathematical descriptions combine precision with understandability in an economical way. Secondly, we can execute the functions and hence validate that they adhere to our informal expectations. It is of course possible to use mathematics to give partial descriptions or to give complete but indirect and hence unexecutable descriptions. For some problems that may be all we can do. But if we can give complete constructive and hence executable descriptions we can make certain and rapid progress in the process of software design. So much so, that we believe the quality of the software and the cost of its production will both be markedly improved.

Of course software design is only part of the software lifecycle. It is preceded by requirements specification about which we have little to say. It is followed by implementation, which if done using traditional imperative languages such as Pascal and C, will benefit enormously from the support of traditional software engineering tools and their integrated future developments, the IPSEs. The techniques which we propose here complement these other phases in that they show how a formal mathematical specification of the software components to be implemented can be achieved and how design alternatives can be investigated. They also show how certainty in the progress of the design can be achieved by members of a design team by executing their constructive mathematical description as a prototype implementation.

The method we shall describe, which is based on functional programming and borrows a great deal from VDM, has been given a name. We call it *me too*. Not for any better reason than it needed a name. We shall use this same label for both the method and the mathematical notation which is part of it.

In normal usage it is clear when one is talking about the method and when one is talking about the mathematics.

We ought not to give the impression that *me too* is as elaborate as VDM[1,3]. It is not. There may be many things which VDM can succinctly describe which *me too* cannot. Learning *me too* however is probably much easier and will serve as a gentle introduction to VDM for those who wish to take formal methods that far.

Finally, before we introduce the method and the mathematics let us explain that this paper only sets out to give an example and leaves much for the interested reader to discover. We use a particular brand of mathematics, where another brand might serve equally well. We use a particular style of prototyping where another might fit better with a different kind of application. We do not wish to purvey these particular products so much as a feeling that the use of formal methods in the design process and some mechanical support for their valid application can greatly aid the software engineer in what remains one of his greatest unsupported activities, that of software design.

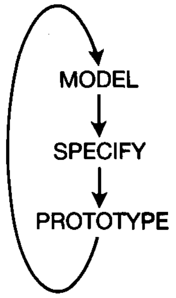
2 The method

The *me too* method is a three step method. The steps are illustrated in Figure 1.

The first two steps, the Model and the Specify steps, concentrate on how to communicate the details of a design among the members of a design team. The third step, the Prototype step, serves to validate those details. These three steps constitute one cycle of the design process. This cycle is repeated, as the diagram indicates, in order to correct design faults or explore alternative design decisions.

The technique used by *me too* to specify a design is similar to that used by VDM. As in VDM, data objects are modelled on familiar mathematical structures, and operations on the objects are formally specified by the natural

Fig. 1 Steps in the *me too* method



mathematical operations allowed on those structures. The *me too* method separates the usual process of formal specification into two steps, the Model step and the Specify step, which are now described.

The first step of the design process, the Model step, is conducted in an informal way by determining the objects (or data structures) involved and the operations upon those abstract objects which are appropriate for the software being designed. Concrete representations are not considered at this point so that the nature of a design is not influenced by concern over 'implementation' details. This first step of the method simply compiles a list of these objects and operations. For each operation we decide those objects upon which it acts and the object it returns. Of course, an informal description is given to each object and each operation.

The list of objects and operations and their descriptions form an abstract model of the design. This model provides a team with a framework in which the members can discuss a design, consider alternative design decisions and generally come to an initial informal agreement about the design. Much of the creative activity of the design process is involved in this step. The explicit attention to enumeration of operations is, we have discovered, an efficient and effective way of capturing those creative design decisions.

At the end of the Model step the design team have reached an informal agreement about the software they have designed. Each object and each operation has been agreed. But, experience shows that this agreement is far from precise. In a traditional design process it may be the highest level of agreement that the design team can reach. However, subsequent experience will undoubtedly demonstrate to them that their design was, in places, flawed.

So, in the Model step a design team propose decisions in an informal way. In the Specify step the members must commit themselves to making these decisions explicit by giving formal meanings to the objects and operations of their design. Before defining the operations, formal representations are given to each object of the model. The objects are represented using the following mathematical structures: sets, relations, finite functions (maps) and sequences. The operations are defined using the natural mathematical operations for the structures on which they act. In *me too*, all operations are defined explicitly, unlike in VDM where some definitions may be implicit. See Reference [4] for a comparison of *me too* and VDM.

The definitions given to the objects and operations in the Specify step transform the informal framework of the model into a rigorous design. The decisions that were loosely formulated in the Model step are precisely expressed in the formal specification. Therefore, at this stage in the process there is a somewhat more complete and common understanding between the members of the team about the details of the design. Typically, in arriving at agreement about the formal specification of objects and operations, the design team will uncover and correct some of the residual design flaws.

The mathematics used in the Specify step aids in the communication of design decisions. It should also assure the correctness of a design. However, although a mathematical specification gives a clear description of a decision, it is still possible that the formal meaning is not the intended meaning. A means for testing a team's understanding of a design is provided in the third step of the method, the Prototype step.

The third step calls for a design to be prototyped according to the formal specification, and for the prototype to be executed in order to validate the design. The operations in a *me too* specification are defined explicitly as constructive functions. The specification is made executable by transcribing the mathematical notation into a functional programming language.

In the Specify step, the design team comes to an understanding as to how the system under design behaves. The Prototype step tests this understanding. The specification is executed and its behaviour is observed. It is found that this step not only discloses misconceptions about design decisions (errors of commission) but also reveals where decisions have been ignored (errors of omission). It is a major contribution of this design method that these failings are discovered early so that the design team has the opportunity to revise the design.

On completion of the third step, the team proceeds to correct errors and fill in holes in the design by repeating the three steps of the method. If the prototype reveals missing operations, the abstract model is extended. If it uncovers errors in existing operations, their formal definitions are amended. The design cycle is repeated in this way until a satisfactory prototype is obtained.

The method encourages a team to build a design incrementally in short cycles so that early feedback, is gained after each decision is made. During any cycle previous decisions may be disregarded to allow investigation of alternative ones. The method is intended to *encourage* iterations during a design. An example of the use of the method to explore different decisions for the design of a simple system for manipulating production rules is given later. First, there is an introduction to the mathematics that the method employs.

3 A little set theory

Of course we can not hope to introduce all of the mathematics which might be required to describe real software in a paper as brief as this one. Jones' book[3] is relied upon for that purpose and you may find it useful to refer to Chapters 7, 8 and 12 which explain some of the mathematics which is not covered in this paper. Jones' paper[1] is an alternative source, but is not a tutorial.

In fact we borrow our set notation from Turner[5,6]. He introduced set notation into a functional language. We expect our sets to behave like the sets of mathematical set theory and have the usual operations of set theory (union, intersection etc) available to us. We will often use a set to denote a data object. For example the set which denotes the binary relation describing those things which certain people possess is written explicitly as

$$\text{possesses} = \{(\text{FRED, BICYCLE}), (\text{TOM, PEN}), (\text{TOM, ROSE}), (\text{BILL, BICYCLE})\}$$

This is a set of pairs. It has four members, each of which is a pair. We use it to denote the data base which records the fact that FRED and BILL possess BICYCLES while TOM possesses a PEN and a ROSE.

Since possesses is a set we can apply various operations into it. So

$$\text{possesses} \cup \{(\text{DICK, STICK})\}$$

is the set which records five possessions, now including DICK's possession of a STICK. And

$$\text{possesses} - \{(\text{TOM, PEN})\}$$

records only three possessions, which are

$$\{(\text{FRED, BICYCLE}), (\text{TOM, ROSE}), (\text{BILL, BICYCLE})\}$$

Well, this is all very familiar set notation. Suppose we want to determine all things which TOM possesses. We write

$$\{t \mid (p, t) \leftarrow \text{possesses}; p = \text{TOM}\}$$

which is read as 'the set of all t, such that (p, t) is drawn from [is a member of] the set possesses and p is equal to TOM.'

This is in fact the set {PEN, ROSE}. Similarly,

$$\{t \mid p \leftarrow \{TOM, BILL\}; (p', t) \leftarrow possesses; p = p'\}$$

denotes the set of all things owned by either TOM or BILL according to the relation possesses. This is the set {PEN, ROSE, BICYCLE}.

Alternatively, we could have written this as

$$\{t \mid (p', t) \leftarrow possesses; p' = TOM \textit{ or } p' = BILL\}$$

We can use sets of this sort to describe the data objects which our software is to manipulate, in a sufficiently rigorous way that the meaning of the operations which we define can be made clear and precise. For example if S is a set of pairs we might define an operation

$$\text{project-right } (S, a) \equiv \{y \mid (x, y) \leftarrow S; x = a\}$$

which determines for each pair (x, y) in S those pairs for which $x = a$ and collects together the second members of each pair.

Hence, project-right (possesses, TOM) = {PEN, ROSE} which is the set of all things that TOM possesses according to the database denoted by the set possesses.

The operation project-right has two arguments. The first is a set of pairs, the second is an atom. The result of this operation is a set of atoms. So we can describe its 'functionality' by the statement

$$\text{project-right: set(Pair)} \times \text{Atom} \rightarrow \text{set(Atom)}$$

This is a mathematical equivalent of a procedure heading. It states that project-right has two arguments and a single result and denotes the types of each of the arguments and of the result.

We shall need various extensions of the basic ideas we have introduced in this section, in order to adequately describe the software which we propose to design as our example, and we shall introduce those extensions as we go along. There is however, one mathematical object that is used a lot in the paper and therefore we shall introduce that here.

The mathematical object that denotes a mapping from objects of one type to another is called a finite function. (VDM calls it a map).

A finite function is the same type as a set of pairs with one distinction. That is, each pair in a finite function has a unique first element.

So, the set

$$\begin{aligned} \text{ismarriedto} = \{&(BILL, MARY), (MARY, BILL), (TOM, SUSAN),\\ &(SUSAN, TOM)\} \end{aligned}$$

is a finite function, but

possesses = {(FRED, BICYCLE), (TOM, PEN), (TOM, ROSE), (BILL, BICYCLE)}

is not. The unrepeated first elements form a set called the domain of the finite function. To obtain the domain of a finite function we use the operation dom. Thus,

dom(ismarriedto) = {BILL, MARY, TOM, SUSAN}

To emphasise that a particular set of pairs is a finite function we write an arrow between the members of each pair instead of a comma and omit the parenthesis. Thus, we would write

ismarriedto ≡ {BILL → MARY, MARY → BILL, TOM → SUSAN, SUSAN → TOM}

This is purely a syntactic device to assist readability of descriptions which use finite functions. A finite function is still a set of pairs and as such can be subjected to all the operations allowed on such sets.

Two other constructs on finite functions are used in this paper. The first is finite function override, which is denoted by the symbol ⊕. It operates on two finite functions and produces a third. It is like set union, except that pairs of the second finite function replace those pairs in the first finite function that have the same domain elements. So, for example,

ismarriedto ⊕ {BILL → JILL, JILL → BILL, MARY → FRED, FRED → MARY}

= {BILL → JILL, MARY → FRED, TOM → SUSAN, SUSAN → TOM, JILL → BILL, FRED → MARY}

The other construct is finite function indexing. Because each domain element of a finite function is mapped to a unique value, we can index it using any element in the domain to obtain the element to which it is mapped.

Thus, to discover who BILL is married to we use the following indexing notation

ismarriedto[BILL]

to get the answer MARY. Notice that this is the familiar notation that is used for arrays in conventional programming languages.

Finally, we can construct finite functions in the following way. Suppose we want to associate each person in the ismarriedto finite function with the things possessed by his or her spouse, according to the possesses relation. We write this association as follows:

$$\{p \rightarrow \text{project-right(possesses, ismarriedto}[p])\mid p \leftarrow \text{dom(ismarriedto)}\}$$

This then constructs the set

$$\{\text{MARY} \rightarrow \{\text{BICYCLE}\}, \text{SUSAN} \rightarrow \{\text{PEN, ROSE}\}, \text{TOM} \rightarrow \{\ \}, \text{BILL} \rightarrow \{\ \}\}$$

None of the uses of the mathematics in this paper is more difficult than that we have already introduced and we hope that the example will be sufficiently interesting that the meaning of the operations will serve to reinforce your understanding of the mathematics.

Bearing in mind our earlier comments about the use of mathematics as a language of communication among the members of a design team, what we hope you will observe in the remaining sections of this paper is the extent to which an elaborate design can be adequately and precisely stated using the mathematical language which we shall introduce.

4 The example

As an illustration of the method we will design a simple system for manipulating production rules similar to that described by Winston[7]. An example of a user of such a system is a manager of a firm. The manager knows of tasks to be performed to which employees of the firm may be allocated. The manager also knows of courses that instruct on skills that employees may lack. The manager provides the system with facts about the requirements of each task, the skills and availability of each employee, and the dates of courses. The manager may capture the way in which he manipulates such information in rules of the form shown in Figure 2.

Fig. 2 An example of production rules

Rule 1	If a task requires a skill and an employee has the skill, then the employee may perform that task.
Rule 2	If an employee may perform a certain task and the employee is available on a certain day, then the employee may be allocated to the task and the task may be performed on that day.
Rule 3	If an employee desires a skill and a course instructs on that skill, then the employee may be interested in the course.
Rule 4	If an employee is interested in a course, and the course is taught on a certain day and the employee is available on that day, then the employee may attend the course on that day.

Each rule has two parts, which we call the 'ifs' part and the 'thens' part. For example, rule 2 has the 'ifs' part

> 'an employee may perform a certain task and the employee is available on a certain day'

and a 'thens' part

> 'the employee may be allocated to the task and the task may be performed on that day'

The production rule system will operate upon a set of given facts and, whenever the facts in the 'ifs' part of a rule match the given facts it will augment those facts with those obtained from the corresponding 'thens' part of the same rule.

For known facts and rules, we require a system which deduces for the manager when and to whom a task may be allocated, and when an employee may attend a course.

Rules of the kind listed in Figure 2 are known in the field of Artificial Intelligence as production rules. Systems that operate on such rules and known facts to deduce other facts are termed production rule systems. The remaining sections of this paper apply the *me too* method of Software Design to produce a first prototype of a production rule system capable of applying rules such as those shown in Figure 2. The design process is described as two major iterations, in the first we only allow constant facts to appear in rules. In the second we elaborate this initial design to include the use of variables in the rules.

5 The first design

5.1 The model-step

The first step of the method is to build an abstract model of the system being designed. This requires us to conduct a preliminary analysis of a production rule system. We borrow the basic characteristics of a production rule system from Winston[7], and from the requirements of the planning application.

Thus we conclude that a production rule system consists of three parts:

(i) a collection of facts,
(ii) a collection of rules,
(iii) a control strategy which operates on the facts and rules.

A particular control strategy will be designed in this paper. It is one that deduces all possible facts from given facts and rules by attempting to match the rules against the facts and repeatedly applying those rules that match successfully. This control strategy is called 'forward-chaining'[7].

The abstract model will reflect each of the three main parts. Consider the first part; the facts.

A model is built for the facts by determining appropriate objects and operations. Two objects are immediately apparent, a collection of facts which Winston called a factbase, and a fact itself. A production rule system must be provided with a factbase therefore operations are needed to set one up. These objects and operations are given in Figure 3. There are two different kinds of object, a Fact and a Factbase. There are two operations. addfact(fb, f) will add Fact f to Factbase fb. The operation emptyfb() creates an initially empty Factbase.

A statement, or signature, is given to denote the type of each operation. In the case of the first operation, the signature states that addfact acts upon objects of type Factbase and Fact and produces an object of type Factbase. The types following the colon and separated by crosses in a signature represent the arguments of an operation, and the type following the arrow represents its result. The second operation takes no arguments, and therefore no types are stated between the colon and arrow in its signature. A description is given with each object and operation to complete this first model for the facts.

Fig. 3 A model for Facts

Objects

Factbase – a collection of facts
Fact – a fact

Operations

addfact: Factbase $\times$ Fact $\rightarrow$ Factbase
addfact(fb,f) adds a fact f to a collection of facts fb.

emptyfb: $\rightarrow$ Factbase
emptyfb() creates an empty collection of facts.

At this point in a design, a design team has some informal understanding of how the operations of the system should behave. The team perhaps simulates the operations by invoking them manually. In the case of this design, the team may consider the following sequence of applications on the operations:

fb = addfact(emptyfb(), f1)
fb′ = addfact(fb, f2)

and hence demonstrate their intention that fb contains the fact f1 and fb′ contains both facts f1 and f2. (Notice, it is not known at this stage exactly how the collection appears, because concrete representations have not yet been assigned to objects). The team must define precisely how they mean these operations to behave in the Specify step. In a practical design situation, the team may feel that there is already enough of a model to proceed through the design cycle and produce the specification and execute it in order to examine the intended behaviour. However, here the model will be extended to cover the remaining two parts that characterise a production rule system.

The next concept to model is the collection of rules which Winston called a rulebase. A rule needs to be labelled with a name, so that a matched rule may be uniquely determined during the matching stage of the control strategy. Therefore, another object is needed to represent a rule name. The system must be provided with the rules, therefore operations are required for establishing a collection of rules. Since a rule name uniquely determines a rule, the decision is made and expressed in the model regarding the addition of a rule to a collection whose name has already been assigned to a rule in that collection. These concepts are described in the model shown in Figure 4.

Fig. 4 A model for rules

Objects

Rulebase	a collection of rules
Rule	a rule
Rulename	a name of a rule

Operations

addrule: Rulebase × Rulename × Rule → Rulebase
addrule(rb,rn,r) adds a rule r with name rn to a collection rb. If a rule already exists in rb under the name rn, then r replaces that rule.

emptyrb: → Rulebase
emptyrb() creates an empty collection of rules.

mkrule: set(Fact) × set(Fact) → Rule
mkrule(ifs, thens) creates a rule from a set of facts representing the 'ifs' part and a set of facts representing the 'thens' part.

ifs: Rule → set(Fact)
ifs(r) gives the set of facts in the 'ifs' part of the rule r.

thens: Rule → set(Fact)
thens(r) gives the set of facts in the 'thens' part of the rule r.

There are three new object types. The operations emptyrb() and addrule(rb, rn, r) are not unlike the operations upon facts. The model also includes operations for constructing and dismantling a rule. The rules for the planning application given in the previous section are characterised as having an 'ifs' part which consists of a set of facts which are matched against the factbase, and a 'thens' part which consists of a set of facts that are deduced when the rule is successfully matched. Operations for creating a rule and selecting the 'ifs' and 'thens' part of a rule are listed respectively.

The final part of the model concerns the control strategy of a production rule system. The strategy for this design contains the following steps.

1 Each rule in the rulebase is matched against facts in the factbase.
2 The name of each rule whose 'ifs' part successfully match facts in the factbase is noted. If there are no such matches then we are finished.

3 For each rule which does match, the facts from the 'thens' of that rule are collected. This new collection of facts is added to the factbase. If all the facts are already there, then we are finished, otherwise we continue from step 1.

The strategy is reflected in the model shown in Figure 5. It is believed that matchapply will apply applyrules to the results of matchrules, which itself performs an application of matchlife. Figure 6 is a diagram of the intended strategy.

Fig. 5 A model for the control strategy

matchrules: Factbase × Rulebase → set(Rulename)
matchrules(fb,rb) returns the set of names belonging to those rules in the collection of rules rb that match facts in the collection fb.

matchifs: Factbase × set(Fact) → Boolean
matchifs(fb,fs) tests whether the facts in fs match facts in the collection fb.

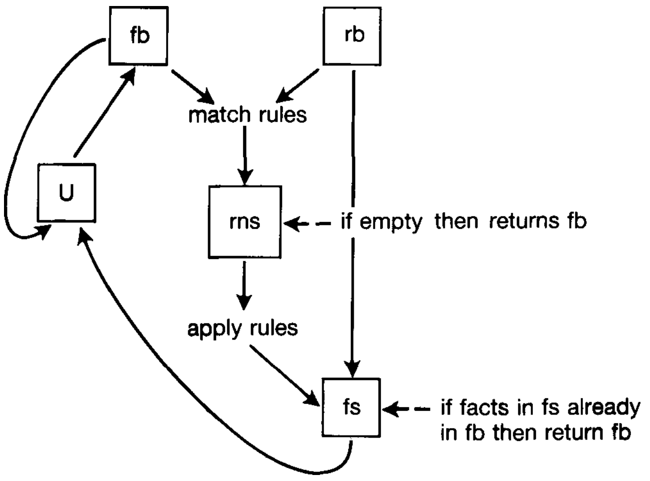
applyrules: Rulebase × set(Rulename) → set(Fact)
applyrules(rb,rns) returns all facts in the 'thens' part of each rule in the collection rb corresponding to a name in the set rns.

matchapply: Factbase × Rulebase → Factbase
matchapply(fb,rb) returns the factbase which results from repeatedly applying the rules in rb to the facts in fb until no further facts can be deduced.

3.2 The specify-step

In building the model, the design team has implicitly decided on aspects of the system's behaviour. However, the team will not yet have conveyed in any of the statements made about the system that it has a total understanding of

Fig. 6 The control strategy



all of these aspects. It is probably the case that the purported understanding varies amongst different members of the team. The team improves its position by using the specification language of *me too* to argue about the details and to come to a complete and common understanding of the design.

Formal meanings are now given to the objects and operations of the abstract model. Consider the model of the facts. The objects, Fact and Factbase, need to be represented by mathematical objects. A fact of the planning system might be the following expression:

task requires skill

An expression can be viewed as a collection of words, or atoms, where the words appear in the collection in a fixed order. The mathematical object that is used to represent an ordered collection of items is a sequence. Another characteristic of a sequence that distinguishes it from a set is that it is allowed to have repeated elements. Thus, it seems appropriate that the object Fact is chosen to be represented as a sequence of atoms. The fact given above appears, then, in its concrete representation as

$\langle$task, requires, skill$\rangle$

The production rule system described in the model operates on an unstructured collection of facts, where facts are not repeated, so a set is used to represent the object Factbase.

The operations on facts can now be defined. The operation addfact takes as arguments a collection of facts fb and a fact f and returns a collection containing f. As fb is a set, the operation simply involves set union. The operation emptyfb has no arguments and returns an empty collection of facts. The empty object whose type is a set is the empty set. These decisions are encapsulated in the specification concerning the facts shown in Figure 7. Notice that the notations set(Fact) and seq(Atom) are used in the definitions of Factbase and Fact. In general, set(T1) denotes a set of objects of type T1 and seq(T2) denotes a sequence of objects of type T2, where T1 and T2 can be any type structured to any level of complexity. In this example, we have in fact an object type that is a set of sequences of atoms.

Fig. 7 A specification for facts

Representations

```
Factbase = set(Fact)
Fact     = seq(Atom)
```

Definitions

addfact: Factbase $\times$ Fact $\rightarrow$ Factbase
addfact(fb,f) $\equiv$ fb $\cup$ {f}

emptyfb: $\rightarrow$Factbase
emptyfb() $\equiv$ *empty*

There is nothing particularly complicated about the mathematics used in this description. We use *empty* to denote the empty set. The expression $\mathrm{fb} \cup \{\mathrm{f}\}$ denotes the set obtained by adding f to the set fb.

Next is the specification of the rules. This is shown in Figure 8. A rulebase associates a rule name with a unique rule. The mathematical object that represents a mapping from an object of type T1 to an object of type T2 is a finite function. We write its type formally as ff(T1, T2). Thus, the representation for the object rulebase is given as

ff(Rulename, Rule)

Continuing now with the specification of the rules, the next definition to provide is the one for adding a rule to a rulebase. Addrule adds a rule r with name rn to a rulebase rb. Thus addrule adds to rb the finite function that maps rn to r. Since the decision was made to replace any existing rule with the name rn in rb with the new rule r, the finite function override construct is used.

The next operation is emptyrb which creates an empty object of type Rulebase. Thus, this operation creates an empty finite function, which is just the empty set.

Now to the representation of a rule. As characterised before, a rule has an 'ifs' part followed by a 'thens' part. The object Rule is therefore a pair.

Fig. 8 A specification for rules

Representations

Rulebase = ff(Rulename, Rule)
Rule = tup(set(Fact), set(Fact))
Rulename = Atom

Definitions

addrule: Rulebase × Rulename × Rule → Rulebase
addrule(rb,rn,r) ≡ rb $\oplus$ {rn → r}

emptyrb: →Rulebase
emptyrb() ≡ *empty*

mkrule: set(Fact) × set(Fact) → Rule
mkrule(ifs, thens) = (ifs,thens)

ifs: Rule → set(Fact)
ifs(r) ≡ first(r)

thens: Rule → set(Fact)
thens(r) ≡ second(r)

The mathematical object that denotes an ordered collection of items which has a fixed size is called a tuple. In general, we write

tup(T1, T2,., Tn)

to denote a tuple of size n, where the first item is of type T1 and whose second is of type T2, and so on, where T1, T2, ..., Tn may be distinct types. Tuples are provided with selector operations. For example, first and second are operations that, when applied to a tuple, return its first and second elements respectively.

The operation mkrule is defined, then, to be the pair whose first element is the set of facts representing its 'ifs' part and whose second is the set of facts representing its 'thens' part. The operations ifs and thens just apply the selector operations first and second.

The final section of the specification defines the control strategy. The strategy is described by four operations. These are shown in Figure 9.

Fig. 9 A specification for a control strategy

```
matchrules: Factbase x Rulebase → set(Rulename)
matchrules(fb,rb) ≡ {rn|(rn, r) ← rb; matchifs(fb,ifs(r))}

matchifs: Factbase x set(Fact) → Boolean
matchifs(fb,fs) ≡ fs ⊆ fb

applyrules: Rulebase x set(Rulename) → set(Fact)
applyrules(rb,rns) ≡ union {thens(rb[rn])|rn ← rns}

matchapply: Factbase x Rulebase → Factbase
matchapply(fb,rb) ≡
    let rns = matchrules(fb,rb)
      if rns = empty then fb
      else let fs = applyrules(rb,rns)
        if fs – fb = empty then fb else matchapply(fb ∪ fs,rb)
```

The Boolean valued operation matchifs(fb,fs) determines whether the fact fs, contained in the 'ifs' part of the rule, is true with respect to (i.e. may be matched successfully to) the current factbase fb. We decide that it does so if every fact in fs is also in fb, that is, if fs is a subset of fb. With this decision we can construct the set of rulenames of rules in the rulebase rb whose 'ifs' match the factbase fb. This is exactly the set

$$\{rn|(rn,r) \leftarrow rb;\ matchifs(fb,ifs(r))\}$$

Hence we have the definition of matchrules(fb,rb).

To apply these matched rules to the current factbase we first calculate all the new facts which will be generated by the 'thens' parts of matched rules.

The set

$$\{thens(rb[rn])|rn \leftarrow rns\}$$

is almost what we want. Each rulename rn in the set of rulenames rns is used to extract the rule rb[rn] from the rulebase rb. The set of facts which make up its 'then' part is constructed. By this means we construct a set of sets of facts and must use *union*, the distributed union operator, to combine these into a single set of rules. This operator behaves as follows:

> If A1, A2, ..., Ak are sets, then
> *union* $\{A1, A2, \ldots, Ak\} = A1 \cup A2 \cup \ldots \cup Ak$.

Finally, we come to the control algorithm itself which is specified in matchapply(fb,rb). Here we have been explicit about the decisions on termination. If an attempt to match the rules in rb against the factbase fb results in an empty set of successfully matched rulenames then matchapply(fb,rb) returns the fb and is finished. Similarly if the set fs = apply thens(rb,rns) is such that no new facts have been constructed, which will be seen if the difference of fs and fb is empty, we return fb and are finished. However, when fs does contain new facts we proceed to construct matchapply(fb ∪ fs, rb) which will continue to apply the rules in rb to a new factbase fb ∪ fs.

The specification is a complete record of the team's understanding of the behaviour of the system. It can be used to dispel any confusion amongst members of the team about any aspect of the system.

3.3 The prototype-step

Although a formal specification is a valuable record of a design, it is not always a manifestation of a correct or complete design. A team's belief of the expected behaviour of a system may not be the one that is actually recorded in the specification, so it is important that in the final step of the method the team has the opportunity to verify its belief and examine whether the decisions it intended are in fact the best decisions.

The specification of the production rule system given above can now be exercised. The operations on facts and rules can be tested. A suitable test might take the following basic rules and facts:

```
rb = {rule1 → ({⟨task, requires, skill⟩,
                 ⟨employee, has, skill⟩},
                {⟨employee, may, perform, task⟩}),
      rule2 → ({⟨employee, may, perform, task⟩,
                 ⟨employee, available, on, day⟩},
                {⟨employee, may, be, allocated, to, task⟩,
                 ⟨task, may, be, performed, on, day⟩}),
      rule3 → ({⟨employee, desires, skill⟩
                 ⟨course, instructs, on, skill⟩},
                {⟨employee, may, be, interested, in, course⟩}),
      rule4 → ({⟨employee, may, be, interested, in, course⟩,
                 ⟨course, taught, on, day⟩,
                 ⟨employee, available, on, day⟩},
                {⟨employee, may, attend, course, on, day⟩})}
```

```
fb = {⟨task, requires, skill⟩,
      ⟨course, instructs, on, skill⟩,
      ⟨course, taught, on, day⟩,
      ⟨employee, available, on, day⟩}
```

The following tests may be made to verify the understanding of the control of the system.

Define fb′ and fb″ as follows.

```
fb′ = addfact(fb,⟨employee, has, skill⟩)
fb″ = matchapply(fb′,rb)
```

The result for fb″ should be

```
fb″ = {⟨task, requires, skill⟩,
       ⟨course, instructs, on, skill⟩,
       ⟨course, taught, on, day⟩,
       ⟨employee, available, on, day⟩,
       ⟨employee, has, skill⟩,
       ⟨employee, may, perform, task⟩,
       ⟨employee, may, be, allocated, to, task⟩,
       ⟨task, may, be, performed, on, day⟩}.
```

Now define fb′ and fb″ to be

```
fb′ = addfact(fb,⟨employee, desires, skill⟩)
fb″ = matchapply(fb″,rb)
```

Then fb″ should be the following collection:

```
fb″ = {⟨task, requires, skill⟩,
       ⟨course, instructs, on, skill⟩,
       ⟨course, taught, on, day⟩,
       ⟨employee, available, on, day⟩,
       ⟨employee, desires, skill⟩,
       ⟨employee, may, be, interested, in, course⟩,
       ⟨employee, may, attend, course, on, day⟩}
```

In fact the prototype we have constructed from the given specification does verify the intent recorded in the specification. It does, however, throw up some controversies about how the system might better behave. For instance, perhaps the system should return only those facts that it deduces. Perhaps it should record the names of the rules that are applied along with the facts that are deduced. These aspects are not explored here. What we shall do is to extend our design to allow rules to contain variables which can match any atoms appearing in a fact.

6 The second design

The second design is a revised version of the first. The major revision is to allow an expression in the 'ifs' part of a rule to match more than one expression in the factbase. This is facilitated by introducing variables into the system. For example, if task and skill are variables then the following expression

task requires skill

will match any of the following expressions:

Wallbuilding requires Bricklaying
Decoration requires Painting
Groupleading requires Management

An expression containing variables will be referred to as a pattern.

In presenting the second design we illustrate how, by the iterative nature of the method, we incrementally modify the model and the specifications given in the first design. We engage repeatedly in the process of revising the model, then revising the specification. At each stage, if it were appropriate we could check the validity of our design decisions by exercising the constructive functions which form their specifications. We omit the details of such testing until the entire second design is complete.

6.1 A model and a specification for variables

Two new objects representing a variable and a pattern are added to the abstract model for the production rule system. The variables in patterns will need to be distinguished in some way. A variable can be represented as a pair whose first item is the atom var and whose second is an atom denoting its name. A pattern is then a sequence of either atoms or pairs of this kind.

The additions to the previous specification are shown in Figure 10.

Here, another object called Value has been introduced, really only to make the specification easier to write. A Value is of type Atom or Variable. Two operations have been defined. The first determines whether an object of type Value is also of type Variable. The second operates on an object that denotes a variable and returns an atom that represents the name of that variable.

6.2 A model and a specification for pattern matching

Some other concepts need to be addressed. The first is pattern matching. The expression given previously is now represented as the Pattern

$\langle$(var, task), requires, (var, skill)$\rangle$

Fig. 10 Variables and Patterns

```
Variable = tup({var},Atom)
Pattern = seq(Value)
Value = Atom ∪ Variable

isvar: Value → Boolean
isvar(v) ≡ not isatom(v) and length(v) = 2 and first(v) = var

name: Variable → Atom
name(v) ≡ second(v)
```

To match this expression against the fact

⟨Wallbuilding, requires, Bricklaying⟩

means associating task to Wallbuilding and skill to Bricklaying. Another object is needed to represent a list that associates variable names to the atoms they match. This object is called an Association, often known as an association list. An operation can now be introduced which attempts to match a Pattern against a Fact. If it is successful it will return an object of type Association. This operation will have the following signature

match: Pattern × Fact → Association

The operation match will not always succeed. We need to be able to distinguish those cases when match will succeed from those when it will not. For this purpose we introduce an operation

matches: Pattern × Fact → Boolean

If matches(p, f) is true then match(p, f) will produce a valid Association. Otherwise match(p, f) is not defined.

Some new decisions are now made about the structure of a rule. The first is that the patterns in the 'ifs' part will be ordered. The second is that the 'thens' part will be also allowed to contain patterns. Thus rule1 in the planning example will now have the following representation.

```
(⟨⟨(var, task), requires, (var, skill)⟩,
  ⟨(var, employee), has, (var, skill)⟩⟩,
 {⟨(var, employee), may, perform, (var, task)⟩})
```

These decisions are accompanied with the one that states that once a variable's name is associated with an atom, that atom will replace all further occurrences of it. Thus each time a variable is matched to an atom, any variable to the right of it sharing its name will be matched to the same atom. So in the above example if the first pattern matches the fact

⟨Wallbuilding, requires, Bricklaying⟩,

then the second pattern will be replaced by the pattern

⟨(var, employee), has, Bricklaying⟩.

If this pattern now matches the fact

⟨Reginald, has, Bricklaying⟩

then the pattern in the 'thens' part will be replaced by the pattern

⟨Reginald, may, perform, Wallbuilding⟩.

These decisions suggest some revisions to the model. The operation mkrule now operates on a sequence of patterns, representing its 'ifs' part, and a set of patterns, representing its 'thens' part. The operations ifs and thens also need the relevant changes, so that the model now has the amended information shown in Figure 11.

Fig. 11 Revised model for rules

```
Fact = Pattern
Rule = tup(seq(Pattern), set(Pattern))
mkrule: seq(Pattern) × set(Pattern) → Rule
ifs: Rule → seq(Pattern)
thens: Rule → set(Pattern)
```

The other revision is an extension to the model. An operation is included that instantiates variables in a pattern with atoms they have previously matched. For example this operation will instantiate the pattern

⟨(var, employee), has, (var, skill)⟩

with the list that associates skill with Bricklaying, and yield the pattern

⟨(var, employee), has, Bricklaying⟩

The operation is called subst (for substitute) and has the following signature

subst: Pattern × Association → Pattern

Notice that sometimes subst returns an expression with variables, whilst on occasion application of it returns one without variables. A restriction will be put on this design, that the use of subst on a pattern in the 'thens' part of the rule will always return a variable-free expression, in other words a fact. So we will not add Facts with variables to the factbase.

Now let us turn to the specification of pattern matching. This is shown completely in Figure 12. Here we have recorded our decision that an Association is a finite function mapping the Atom representing the name of a

Fig. 12 Pattern matching

```
Association = ff(Atom, Atom)

subst: Pattern × Association → Pattern
subst(p, a) ≡ ⟨if isvar(v) and name(v)∈dom(a)
  then a[name(v)] else v|v←p⟩

match: Pattern × Fact → Association
match (p, f) ≡ {name(p[i]) → f[i]|i←dom(p); isvar(p[i])}

matches: Pattern × Fact → Boolean
matches(p, f) ≡
  len(p) = len(f) and
    and/{isvar(p[i]) and
        and/{f[i] = f[j]|j←dom(p);
            isvar(p[j]) and p[i] = p[j]}
        or not isvar(p[i]) and p[i] = f[i]|i←dom(p)}
```

variable to the Atom representing its value. The meaning of subst(p, a) is then clearly expressed as the construction of a Pattern (a sequence) congruent to the sequence p but with those variables whose value is defined by a replaced by the corresponding value. The definition of match(p, f) takes advantage of the fact that a sequence is also a finite function whose domain is a set of integers {1, ..., len(p)}. Note that match(p, f) is only defined when the fact f does match the pattern p, in which case the Association is constructed which maps each variable appearing in p to the corresponding value appearing in f.

All of the hard work has been left to the Boolean valued operation matches(p, f). In this we use a peculiar device which we write *and*/{...} where the set is a set of Boolean values. This expression is true only when all of the elements in the set is true. In mathematical logic such an expression might more conventionally be written using a universal quantifier. The body of matches(p, f) can be read as follows.

(i) p and f are of the same length

and

(ii) For every element p[i] of the pattern p
either (a) p[i] is a variable and for every other occurrence p[j] of the same variable the same value is matched in the fact f (f[i] = f[j])
or (b) p[i] is not a variable in which case it is identical to f[i].

Now this may not be the simplest form of expression for these three operations. It was not the first form we wrote down, nor the first form which we took to prototype. On reconsidering our first specification, which we thought unnecessarily complex, we discovered the specification written in Figure 12. We were convinced that it correctly conveyed our meaning, and so it proved on testing in prototype. There was a minor slip, one of the *and*/{...} operators was omitted, but this omission was easily detected and rectified. The specification had served its main purpose of being a precise vehicle of communication between us as designers – we used it to argue about our intuitions, to efficiently offer alternatives to each other and eventually to document our decisions to those of you who will seek its exact meaning. No doubt we will eventually discover a simpler expression of this meaning. No doubt we will discover residual design flaws or nuances which we had not considered. If so, and particularly if these discoveries involve a third party, then again the specification will have served its main purpose – to communicate.

6.3 *A model and a specification for the control strategy*

First, the model-step. The operations for the control strategy need to be revised. Recall the operations for matching rules, matchrules and matchifs. The operation matchifs took a factbase and a set of facts, representing the 'ifs' part of a rule, and returned a boolean stating whether or not the facts in the set successfully matched facts in the factbase. The operation now takes a factbase and a sequence of patterns as arguments and returns, not a boolean, but a set of association lists for the variables it has matched. It returns a set because the sequence of patterns can match the factbase in more than one way. Take, for example, rule3 shown in Figure 13 (the Planning Example), which has the following sequence of patterns in its 'ifs' part.

Fig. 13 The Planning Example

```
rule1→(⟨⟨(var, task), requires, (var, skill)⟩,
        ⟨(var, employee), has, (var, skill)⟩⟩,
       {⟨(var, employee), may, perform, (var, task)⟩})
rule2→(⟨⟨(var, employee), may, perform, (var, task)⟩,
        ⟨(var, employee), available, on, (var, day)⟩⟩,
       {⟨(var, employee), may, be, allocated, to, (var, task)⟩
        ⟨(var, task), may, be, performed, on, (var, day)⟩}),
rule3→(⟨⟨(var, employee), desires, (var, skill)⟩,
        ⟨(var, course), instructs, on, (var, skill)⟩⟩,
       {⟨(var, employee), may, be, interested, in, (var, course)⟩})
rule4→(⟨⟨(var, employee), may, be, interested, in, (var, course)⟩,
        ⟨(var, course), taught, on, (var, day)⟩,
        ⟨(var, employee), available, on, (var, day)⟩⟩,
       {⟨(var, employee), may, attend, (var, course), on, (var, day)⟩})
```

⟨⟨(var, employee), desires, (var, skill)⟩,
 ⟨(var, course), instructs, on, (var, skill)⟩⟩

The sequence can match the following facts from the factbase shown in Figure 14.

⟨Pamela, desires, Carpentry⟩
⟨Reginald, desires, Typing⟩
⟨Barbara, desires, Therapy⟩
⟨Woodwork, instructs, on, Carpentry⟩
⟨Secretarial, instructs, on, Typing⟩
⟨Psychology, instructs, on, Therapy⟩

to produce the following set of association lists:

{{employee → Reginald, skill → Typing, course → Secretarial}
{employee → Pamela, skill → Carpentry, course → Woodwork}
{employee → Barbara, skill → Therapy, course → Psychology}}

The operation matchrules previously took a factbase and a rulebase as arguments and returned a set of names belonging to rules that are successfully matched. It still takes those arguments, but now it returns with each rule name the set of association lists resulting from the match. In other words, it returns a mapping from rule names to the sets of association lists the corresponding rule generates.

Thus, the new signatures for the two operations are as follows:

matchrules: Factbase × Rulebase → ff(Rulename, set(Association))
matchifs: Factbase × set(Pattern) → set(Association)

Fig. 14 A sample factbase

```
{⟨Barbara, has, Management⟩,
 ⟨Reginald, has, Carpentry⟩,
 ⟨Reginald, has, Bricklaying⟩,
 ⟨Reginald, has, Painting⟩,
 ⟨Pamela, has, Typing⟩,
 ⟨Pamela, has, Dictation⟩,
 ⟨Pamela, has, Wordprocessing⟩,
 ⟨Woodwork, instructs, on, Carpentry⟩,
 ⟨Woodwork, taught, on, Thursday⟩,
 ⟨Secretarial, instructs, on, Typing⟩,
 ⟨Secretarial, instructs, on, Dictation⟩,
 ⟨Secretarial, taught, on, Monday⟩,
 ⟨Secretarial, taught, on, Wednesday⟩,
 ⟨Psychology, instructs, on, Therapy⟩,
 ⟨Psychology, taught, on, Tuesday⟩,
 ⟨Pamela, desires, Carpentry⟩,
 ⟨Reginald, desires, Typing⟩,
 ⟨Barbara, desires, Therapy⟩,
 ⟨Clerical, requires, Typing⟩,
 ⟨Clerical, requires, Dictation⟩,
 ⟨Decoration, requires, Painting⟩,
 ⟨Barbara, available, on, Monday⟩,
 ⟨Barbara, available, on, Tuesday⟩,
 ⟨Barbara, available, on, Wednesday⟩,
 ⟨Barbara, available, on, Thursday⟩,
 ⟨Barbara, available, on, Friday⟩,
 ⟨Pamela, available, on, Wednesday⟩,
 ⟨Pamela, available, on, Thursday⟩,
 ⟨Pamela, available, on, Friday⟩,
 ⟨Reginald, available, on, Monday⟩,
 ⟨Reginald, available, on, Wednesday⟩,
 ⟨Reginald, available, on, Friday⟩}
```

Now that matchrules returns something different, the operation for applying rules must be altered. Instead of applyrules taking a set of rulenames as an argument, it now takes a finite function from rule names to sets of association lists. This implies that it must do something with the association lists. For each rule, it must instantiate the 'thens' part with each association list generated.

This will be achieved by defining another operation called applythens, which applyrules will use, which takes as arguments a set of patterns and an association list, and will use the association list to instantiate all the patterns in the set. Because the decision was made that all patterns in the 'thens' part of a rule will be fully instantiated, this operation will return a set of facts. Then the operation, applyrules, that calls it will also return a set of facts, which it did previously. The two signatures for these operations are:

applyrules: Rulebase $\times$ ff(Rulename, set(Association)) $\rightarrow$ set(Fact)
applythens: set(Pattern) $\times$ Association $\rightarrow$ set(Fact)

The operation that performs the overall control retains the same signature. Recall that it had the following signature:

matchapply: Factbase $\times$ Rulebase $\rightarrow$ Factbase

Now to the Specify step, again. The first operation to define is matchrules. It takes each rulename-rule pair (rn, r) from the rulebase rb and calls matchifs on the factbase fb, and the 'ifs' part of r. It is interested in successful matches only. It knows that a match is successful if matchifs returns a non-empty set of association lists. Thus, for each rule r where matchifs(fb, ifs(r)) is a non-empty set, matchrules maps its name to that set. Its definition, then, is as follows:

$$\{\mathrm{rn} \rightarrow \mathrm{matchifs(fb, ifs(r))} \mid (\mathrm{rn, r}) \leftarrow \mathrm{rb}; \quad \mathrm{matchifs(fb, ifs(r))} \neq \mathit{empty}\}$$

The next definition is that of matchifs. This is now more complicated. Matchifs matches every pattern in the sequence in turn. For each pattern, it must take into account previous associations between variables and atoms that have been generated whilst matching earlier patterns. For example, with the sequence of patterns in the 'ifs' part of rule3, when matching the second pattern matchifs must take into account the associations

{employee $\rightarrow$ Reginald, skill $\rightarrow$ Typing}
{employee $\rightarrow$ Pamela, skill $\rightarrow$ Carpentry}
{employee $\rightarrow$ Barbara, skill $\rightarrow$ Therapy}

that were generated whilst matching the first pattern. Consider, in detail, what matchifs will do when applied to this particular example:

When matchifs matches the first pattern

$\langle$(var, employee), desires, (var, skill)$\rangle$

it has no previous association lists to worry about. It must use the operations matches and match to match the pattern against facts in the factbase. But matches and match only match a pattern against one fact. Thus, another operation is needed.

This new operation is called matchpattern. It has the following signature and definition.

matchpattern: Factbase $\times$ Pattern $\rightarrow$ set(Association)
matchpattern(fb, p) $\equiv$ $\{$match(p, f)$|$f $\leftarrow$ fb;
matches(p, f)$\}$

It takes a factbase fb and a pattern p, and attempts to match p to every fact f in fb using the operation match. When matches(p, f) is true then the Association computed by match(p, f) is collected. It therefore returns the set of association lists that represent the successful matches of p to facts in fb.

Consider now how matchifs must process the 'ifs' part of rule3. It applies matchpattern to the first pattern to produce the set

$\{\{$employee $\rightarrow$ Reginald, skill $\rightarrow$ Typing$\}$,
$\{$employee $\rightarrow$ Pamela, skill $\rightarrow$ Carpentry$\}$,
$\{$employee $\rightarrow$ Barbara, skill $\rightarrow$ Therapy$\}\}$

Call this set A1.

It next turns to the second pattern

$\langle$(var, course), instructs, on, (var, skill)$\rangle$

Now it must take into account the set A1. It must apply subst to the pattern with every association list in A1 to produce a new pattern.

For example, if the association list

$\{$employee $\rightarrow$ Reginald, skill $\rightarrow$ Typing$\}$

is taken, the pattern becomes

$\langle$(var, course), instructs, on, Typing)$\rangle$

This pattern is now matched against fb using matchpattern to produce the following set of association lists:

$$\{\{\text{course} \rightarrow \text{Secretarial}\}\}$$

In general of course this set of Associations will have more than one element. Each Association in this set will be added to the Association which instantiated the pattern. This process will be repeated for each of the Associations in A1 to produce the set A2. When the 'ifs' part of rule contains n patterns pl, ..., pn we will push a set of Associations through these patterns. Each set of Associations Ak is used to instantiate pattern p(k + 1) which is matched against the factbase and eventually produces the (possibly empty) set of Associations A(k + 1). We can express this relationship by defining a function

$$\text{cascade: Factbase} \times \text{Pattern} \times \text{set(Association)} \rightarrow \text{set(Association)}$$

so that

$$A(k+1) = \text{cascade}(fb, p(k+1), Ak)$$

The definition of cascade is straight-forward:

$$\text{cascade}(fb, p, A) \equiv \{a \oplus a' \mid a \leftarrow A;\ a' \leftarrow \text{matchpattern}(fb, \text{subst}(p, a))\}$$

Matchifs uses cascade on each of the patterns in the sequence it is matching. For the particular example it is being applied to, it performs the following steps.

It defines the set A1 to be

$$A1 = \text{cascade}(fb, \langle(\text{var, employee}), \text{desires}, (\text{var, course})\rangle, \{\textit{empty}\})$$

to get

$$\begin{aligned} A1 = \{&\{\text{employee} \rightarrow \text{Reginald, skill} \rightarrow \text{Typing}\} \\ &\{\text{employee} \rightarrow \text{Pamela, skill} \rightarrow \text{Carpentry}\}, \\ &\{\text{employee} \rightarrow \text{Barbara, skill} \rightarrow \text{Therapy}\}\} \end{aligned}$$

It then defines A2 to be

$$A2 = \text{cascade}(fb, \langle(\text{var, course}), \text{is, taught, on}, (\text{var, day})\rangle, A1)$$

to get

$$\begin{aligned} A2 = \{&\{\text{employee} \rightarrow \text{Reginald, skill} \rightarrow \text{Typing, course} \rightarrow \text{Secretarial}\} \\ &\{\text{employee} \rightarrow \text{Pamela, skill} \rightarrow \text{Carpentry, course} \rightarrow \text{Woodwork}\} \\ &\{\text{employee} \rightarrow \text{Barbara, skill} \rightarrow \text{Therapy, course} \rightarrow \text{Psychology}\} \end{aligned}$$

In general we compute

$$\begin{aligned}
&\mathrm{A0} = \{\mathit{empty}\}\\
&\mathrm{A1} = \mathrm{cascade(fb, pl, A0)}\\
&\vdots\\
&\mathrm{An} = \mathrm{cascade(fb, pn, A(n-1))}
\end{aligned}$$

When we have a series of this form where, for some function f: X × Y → Y and some sequence = ⟨x1, ..., xn⟩ and wish to compute yn defined by

$$\begin{aligned}
&\mathrm{y1} = \mathrm{f(x1, y0)}\\
&\vdots\\
&\mathrm{yn} = \mathrm{f(xn, y(n-1))}
\end{aligned}$$

we write

$$\mathrm{yn} = \mathrm{reduce}(\mathrm{y0}, \lambda(\mathrm{x,y}).\mathrm{f(x,y)}, \langle \mathrm{x1}, \ldots, \mathrm{xn}\rangle).$$

Thus, using this notation, matchifs applied to a factbase fb and a sequence of patterns pq has the following definition:

$$\mathrm{matchifs(fb, pq)} \equiv \mathrm{reduce}(\{\mathit{empty}\}, \lambda(\mathrm{p,A}).\mathrm{cascade(fb, p, A), pq})$$

That completes the specification for matching rules. Notice two new operations, cascade and matchpattern had to be introduced in order to specify matchifs.

Now the specification for applying rules is given. The two operations to specify are applyrules and applythens. Applyrules has the following definition:

Fig. 15 The revised control strategy

```
matchrules: Factbase × Rulebase → ff(Rulename, Association)
matchrules(fb, rb) ≡ {rn → matchifs(fb, ifs(r)) | (rn, r) ← rb; matchifs(fb, ifs(r)) ≠ empty}

matchifs: Factbase × seq(Pattern) → set(Association)
matchifs(fb, pq) ≡ reduce({empty}, λ(p,A).cascade(fb, p, A), pq)

cascade: Factbase × Pattern × set(Association) → set(Association)
cascade(fb, p, A) ≡ {a ⊕ a' | a ← A; a' ← matchpattern(fb, subst(p, a))}

matchpattern: Factbase × Pattern → set(Association)
matchpattern(fb, p) ≡ {match(p, f) | f ← fb; matches(p, f)}

applyrules: Rulebase × ff(Rulename, set(Association)) → set(Fact)
applyrules(rb, rns) ≡ union {applythens(thens(rb[rn]), a) | rn ← dom(rns); a ← rns[rn]}

applythens: set(Pattern) × Association → set(Fact)
applythens(ps, a) ≡ {subst(p, a) | p ← ps}

matchapply: Factbase × Rulebase → Factbase
matchapply(fb, rb) ≡
     {let rns = matchrules(fb, rb)
      if rns = empty then fb
      else let fs = applyrules(rb, rns)
        if fs − fb = empty then fb else matchapply(fb ∪ fs, rb)}
```

$$\text{applyrules(rb, rns)} \equiv$$
$$\mathit{union}\{\text{applythens(thens(rb[rn]), a)} \mid \text{rn} \leftarrow \text{dom(rns); a} \leftarrow \text{rns[rn]}\}$$

It takes a rulebase rb and a finite function rns which matches rulenames to sets of association lists. It takes a rulebase rb and every rulename rn

Fig. 16 Results from the Prototype

```
{⟨Barbara, has, Management⟩,
 ⟨Reginald, has, Carpentry⟩,
 ⟨Reginald, has, Bricklaying⟩,
 ⟨Reginald, has, Painting⟩,
 ⟨Pamela, has, Typing⟩,
 ⟨Pamela, has, Dictation⟩,
 ⟨Pamela, has, Wordprocessing⟩,
 ⟨Woodwork, instructs, on, Carpentry⟩,
 ⟨Woodwork, taught, on, Thursday⟩,
 ⟨Secretarial, instructs, on, Typing⟩,
 ⟨Secretarial, instructs, on, Dictation⟩,
 ⟨Secretarial, taught, on, Monday⟩,
 ⟨Secretarial, taught, on, Wednesday⟩,
 ⟨Psychology, instructs, on, Therapy⟩,
 ⟨Psychology, taught, on, Tuesday⟩,
 ⟨Pamela, desires, Carpentry⟩,
 ⟨Reginald, desires, Typing⟩,
 ⟨Barbara, desires, Therapy⟩,
 ⟨Clerical, requires, Typing⟩,
 ⟨Clerical, requires, Dictation⟩,
 ⟨Decoration, requires, Painting⟩,
 ⟨Barbara, available, on, Monday⟩,
 ⟨Barbara, available, on, Tuesday⟩,
 ⟨Barbara, available, on. Wednesday⟩,
 ⟨Barbara, available, on, Thursday⟩,
 ⟨Barbara, available, on, Friday⟩,
 ⟨Pamela, available, on, Wednesday⟩,
 ⟨Pamela, available, on, Thursday⟩,
 ⟨Pamela, available, on, Friday⟩,
 ⟨Reginald, available, on, Monday⟩,
 ⟨Reginald, available, on, Wednesday⟩,
 ⟨Reginald, available, on, Friday⟩,
 ⟨Pamela, may, perform, Clerical⟩,
 ⟨Reginald, may, perform, Decoration⟩,
 ⟨Pamela, may, be, allocated, to, Clerical⟩,
 ⟨Clerical, may, be, performed, on, Wednesday⟩,
 ⟨Clerical, may, be, performed, on, Thursday⟩,
 ⟨Clerical, may, be, performed, on, Friday⟩,
 ⟨Reginald, may, be, allocated, to, Decoration⟩,
 ⟨Decoration, may, be, performed, on, Monday⟩,
 ⟨Decoration, may, be, performed, on, Wednesday⟩,
 ⟨Decoration, may, be, performed, on, Friday⟩,
 ⟨Pamela, may, be interested, in, Woodwork⟩,
 ⟨Reginald, may, be, interested, in, Secretarial⟩,
 ⟨Barbara, may, be, interested, in, Psychology⟩,
 ⟨Pamela, may, attend, Woodwork, on, Thursday⟩,
 ⟨Reginald, may, attend, Secretarial, on, Monday⟩,
 ⟨Reginald, may, attend, Secretarial, on Wednesday⟩,
 ⟨Barbara, may, attend, Psychology, on, Tuesday⟩}
```

contained in rns, that is every name in the domain of rns, and each association list a to which rn maps in rns and applies applythens to the 'thens' part of the rule associated with rn in rb.

Applythens, remember, returns a single sets of facts. Thus applyrules is collecting a set of set of facts. But applyrules must also return just a single set of facts. Thus it must apply the distributed union operator to the set of sets.

Finally, applythens has the following definition

$$\text{applythens}(ps, a) = \{\text{subst}(p, a) \mid p \leftarrow ps\}$$

It takes a set of patterns ps and applies the operation subst to each pattern in ps. The entire design is collected together in Figure 15.

The overall control operation matchapply is the same as in the first design. For completeness, it is given with the complete specification of the revised control strategy.

6.4 The prototype-step

We now turn the specification into a prototype and exercise it. We might set up the rulebase and factbase given in Figures 13 and 14. We then define

$$fb' = \text{matchapply}(fb, rb)$$

We would expect to find that fb′ is the set contained in Figure 16. Of course other extensive tests should be performed and were performed by us. The production system did behave in the way in which we intended, but when we played with it we became aware of some more sophisticated facilities which it lacked and to which we then went on to address ourselves. Neither space nor your patience will allow us to describe those later developments here.

5 Conclusions

We have introduced a method of software design which yields a formal mathematical description of the design and a demonstration of the validity of that specification in the form of an executable prototype. We have tried to convey our belief that this approach to software design can produce the executable prototype very rapidly. The prototype is not intended to be the implementation: that must be produced by more conventional means, at which time attention will be paid to economical use of machine resources. Eventually the advent of more powerful computers and more sophisticated methods of implementation for functional languages may change this situation. Nevertheless, considered as a part of the traditional software development process, the rapid production of a validated, formal specification of the design, can, we believe, contribute significantly to improvements in software quality and to the cost of software production. Firstly the method concentrates on the iterative nature of design and on the need for communication of design decisions among the members of a design team. The Model-step encourages the team to investigate alternative designs and gives them a vocabulary (the names of the objects and the operations) to facilitate their early informal design discussions. The agreements which the team reaches at this stage are informal, being captured in English descriptions of the components of the model.

The Specify-step forces the team to make precise statements of the meaning of every object and every operation in the model. Typically at this stage the team discovers inadequacies in the design, which at the Model-step had not appeared. For example they discover that some cases had not been considered and some design decisions not made. They also discover that the understanding of each team member differs in places from the understanding of the others. The apparent agreement reached at the Model-step is improved now by the precise statement of the new agreement, reached once these discrepancies of understanding are resolved.

The design could of course be considered complete at this stage. All of the creative work is done. Experience shows us however that the kinds of mistakes which one traditionally makes in programming are also made in specification. So at the Prototype-step we turn our specification into an executable prototype of the software and carry out a traditional testing process. This invariably uncovers errors of omission and errors of commission in the specification. It forces us to an even higher level of precision and agreement. It also encourages us to consider alternatives, for often we discover that the design we have made makes certain requirements awkward or difficult to achieve and testing the prototype suggests improvements to the design over and above those suggested in the earlier stages of the method.

So iteration takes place at all stages of the design. Whether we can afford to carry out many repeated redesigns is very dependent upon the cost of applying the method. It is our belief that the effort involved in producing a formal specification and a prototype to validate it is modest in the context of the entire cost of production of a real software product. If indeed this is the case and if the early availability of a clear, precise formal description of the design reduces the traditionally high cost of late discovery of design errors in the software, there is some basis for our belief that a method such as that described here can contribute significantly to the production of low-cost, high-quality software. However, this is not yet the result of experimental evidence. The project in which we are now engaged, supported by ICL and by the SERC through the Alvey directorate, is attempting to industrialise this method, which of course involves both its application and its development.

The mathematics we have used is borrowed largely from VDM. We have chosen a subset of that notation which is executable as a functional program. We conjecture that it is powerful enough to describe any software architecture with some succinctness. We could have chosen OBJ which has a similar executable subset[8] or the specification language Z which is the first one we studied[9]. Our choice was largely governed by the availability of Jones's book[3] at the time we began this work.

The development described here in fact went on to other more elaborate designs of production rule systems. We are currently engaged in an attempt to combine the more elaborate of these designs with a design of a Frames database, partly as an exploration of potential applications for those pieces of software, but mainly as an experiment in building from reusable software components.

Our purpose here has been to encourage you to consider using our method in your own software design tasks and to consider the serious use of mathematics as a language for capturing designs and as a language for communication with your co-designers. The paper was not intended as a tutorial. You will need to explore further the ideas introduced here and the best way to do that is to try to apply them yourself. You will then find that the additional material available in the references is able to answer the questions that arise and that you will be able to adapt the method to your own kind of problem and begin to develop it.

References

1 BJORNER, D. and JONES, C.B.: 'Formal Specification and Software Development.' Prentice Hall International, Series in Computer Science. ISBN 0 13 329003 4, 1981.
2 HENDERSON, P.: '*me too* – a language for software specification and model building – preliminary report' FPN-9, Computing Science Department, University of Stirling, 1984.
3 JONES. C.B.: 'Software Development: A rigorous Approach.' Prentice Hall International, Series in Computer Science. ISBN 13 331579 7, 1980.
4 MINKOWITZ, C.: 'Specification to Prototype – a comparison of two formal methods of software design' TR.18, Computing Science Department, University of Stirling, 1985.
5 DARLINGTON, J., HENDERSON, P. and TURNER, D.A. (eds.): 'Functional Programming and its applications – An Advanced Course', Cambridge University Press, ISBN 0 521 24503 6, 1982.
6 TURNER, D.A.: 'Functional Programs as Executable Specifications', in Mathematical Logic and Programming Languages, C.A.R. Hoare and J.C. Sheperdson (Eds.), 1985.
7 WINSTON, P.H. and HORN, B.K.P.: 'LISP' Addison-Wesley Publishing Company, ISBN 0 201 08329 9, 1981.
8 GOUGEN, J. and MESEGUER, J.: 'Rapid Prototyping in the OBJ Executable specification Language' ACM Sigs of Software Engineering Notes 7(5), p. 75, 1982.
9 SUFRIN, B.: 'Towards a formal specification of the ICL Data Dictionary', ICL Technical Journal, pp. 195–217, Nov. 1984.

Formal specification — a simple example

D.A. Duce and E.V.C. Fielding
Rutherford Appleton Laboratory, Didcot, Oxfordshire

Abstract

The Graphical Kernel System (GKS) is now an ISO International Standard for computer graphics programming. GKS is currently described in an informal way; this paper presents some early results from a project which is looking at the applicability of formal techniques to the problem of specifying GKS. A specification of a simplified model of GKS is progressively constructed using a subset of the Vienna Development Method (VDM) of formal specification. This illustrates how rich specifications can be obtained from simpler ones. GKS concepts and VDM notation are described as the development proceeds.

1 Introduction

1.1 Background

Standards for computer graphics programming have lagged far behind standards for programming languages. *De facto* standards existed in programming languages from the very early days, e.g. Fortran and Algol 60, and international standards soon followed[1]. However, in the graphics area a different pattern has emerged despite the fact that the origins of computer graphics can be traced back almost to the birth of digital computers. This was because it was not until the advent of storage tube technology in the late 1960s that hardware costs reduced to a level at which graphics systems were widely affordable. In recent years the cost of raster graphics displays has decreased dramatically and this, coupled with the emergence of high-performance single user workstations, such as the ICL PERQ, is causing a further revolution in the role of graphics and graphical interfaces in system design.

A major problem facing system designers is how to write applications programs which are portable across a range of graphics devices without requiring major rewriting. This is one of the principal concerns that has motivated activities to develop standards for computer graphics.

The history of graphics standardisation has been recorded elsewhere[2].

GKS[3] is an important landmark in the development of computer graphics, because it is the first system to be produced as an international standard. The signs are that it will be an important software standard for the future; many computer manufacturers, graphics equipment manufacturers and software houses are offering GKS implementations or GKS-based products. ICL offer a GKS implementation on the PERQ, which was developed in collaboration with Rutherford Appleton Laboratory.

1.2 GKS

GKS is a two-dimensional graphics system which aims to provide an interface between application programs and a wide variety of graphics devices. It is defined independently of programming languages, although standard bindings to programming languages (e.g. Fortran and Pascal) are being developed. GKS caters for both graphical output and graphical input. A full description of GKS is contained in Reference 2.

The document describing GKS is some 245 pages long. Despite strenuous efforts by the ISO working group and the editors of the document it is inevitable in a document of this size that there will be ambiguities and omissions; places where a knowledge of the review process is necessary to fully understand what is required of an implementation. The issue of validation of GKS implementations is a crucial area, presently being addressed.

The need for rigorous specification of software products is gradually being recognised in the software industry and it is our belief that the most rigorous specifications possible should be given for standards, if the word 'standard' is to have any meaning.

Previous articles in this journal have looked at the application of formal techniques to the specification of protocols[4] and databases[5]. Graphics is an area to which formal techniques have not as yet been widely applied. It is our belief that there is an urgent need for work in this area, to discover ways of presenting complex standards in a clear, readable and unambiguous form.

The work described in this paper has arisen during the early stages of a research project being carried out by the authors at the Rutherford Appleton Laboratory to explore the application of formal specification techniques to GKS. The project is funded by the Alvey Directorate as part of its Software Engineering Programme and ICL West Gorton act as the project's Alvey Uncle.

The project is proceeding by looking at the application of formal techniques to small parts of GKS. The parts are being chosen on the basis of our understanding of GKS that they will be key components in any complete specification. The approach is thus to establish the viability of particular techniques and styles of using them on small systems before embarking on a more complete specification.

The small system described here has been described from a different viewpoint[6], where the emphasis was on how one might reason about the behaviour of a system from its specification. This paper concentrates on illustrating how rich specifications may be obtained from simpler ones, by using this example to take the reader step by step through its construction. Each step introduces and captures a small number of GKS concepts and these concepts are explained informally as the development proceeds.

The specification technique used in this paper is based on the Vienna Development Method (VDM)[7]. Only a limited subset of VDM and a simplified notation have been used.

2 Formal specification technique

The purpose of a specification is to state **what** a system is to do, not **how** it is to do it. It does this by describing the internal state of the system in an implementation-independent way by the use of **abstract data types**. An abstract data type is characterised only by the operations allowed over it and is representation-independent.

The specification technique used in this paper, VDM, is an example of the **constructive** or **model-based** approach, which models abstract data types in terms of mathematically tractable entities such as basic types (integers, reals etc.) and tuples, sets, lists and mappings.

The VDM specifications given in this paper consist of:

- a model of the state
- operations over the abstract data type comprising the state, which characterise it

The state definition describes the structure of the class of objects representing the state in terms of basic and constructed types. The operations are defined implicitly by predicates, which allows relations and thus nondeterminacy to be specified. However the operations given in this paper do not require this generality, and in fact reduce to functions. Operation definitions given here then have the general type:

$$\mathit{State} \times \mathit{Inputs} \rightarrow \mathit{State}$$

and are described by two predicates: a **precondition** and a **postcondition**. The former is a predicate over *State* and *Inputs* and defines the conditions under which the operation produces a valid result. The latter is a predicate over *State* (the initial state), *Inputs* and *State* (the final state), which defines the effect of the operation.

3 The first specification

3.1 Basic GKS concepts

GKS provides a functional interface between an application program and a configuration of graphical input and output devices at a level of abstraction that hides the peculiarities of device hardware. It achieves device independence by means of the concepts of abstract input, abstract output and abstract workstations. The concept of abstract input will not be discussed, but the specification attempts to capture the other two concepts.

First, what is meant by abstract output? In GKS, pictures are constructed from a number of basic building blocks, called **output primitives**, which are abstractions of the basic actions that a graphical output device can perform (e.g. drawing a line). There are six output primitives in GKS: polyline, polymarker, text, fill area, cell array and generalised drawing primitive (GDP); each of which has associated with it a set of **parameters**, which define a particular instance of the primitive. This paper considers the polyline primitive, which draws a connected sequence of line segments and has the co-ordinates of its vertices as parameters.

The concept of an abstract **workstation** in GKS is an abstraction from physical device hardware and maps physical output primitives to abstract output primitives and physical input primitives to abstract input primitives. It represents zero or one display surfaces and zero or more input devices as a configuration of abstract devices. An application program may direct output to more than one workstation simultaneously; however, the specification will aim to model only a system with a single workstation.

The application program specifies co-ordinate data in the parameters of an output primitive in **world co-ordinates** (WC), a Cartesian co-ordinate system. World co-ordinates are then transformed to a uniform co-ordinate system for all workstations, called **normalised device co-ordinates** (NDC) by a window to viewport mapping termed a **normalisation transformation.** A second window to viewport mapping, called the **workstation transformation**, accomplishes the transformation to the **device co-ordinates** (DC) of the display surface.

To simplify the specification, WC co-ordinate space will be ignored and it will be assumed that polyline co-ordinate data are supplied in NDC co-ordinates. It is also assumed that the single workstation transformation is fixed.

This description of GKS has extracted the basic concepts of a complex system. These can now be described formally to form a framework on which to build three more specifications which successively capture more of the concepts and complexity of the real system. The next section gives the formal specification of a system embodying the basic concepts.

3.2 The basic specification

The state and operations of a formal specification of the simplified GKS system described above must capture the concepts of NDC space, a workstation and the GKS polyline function.

These concepts are captured in the GKS state by having one component which models the NDC picture and a second component which models the concept of a workstation. In this simplified system, a workstation can be represented simply by a model of the DC picture that is displayed on the display surface. The polyline function is captured by an operation, *add_ polyline*, defined over the state which gives the relationship between these two pictures and shows how they are constructed.

This leads to the following definition for the state of the system.

The state

GKS = *NDC_Picture* × *DC_Picture*

NDC_Picture = **list of** *Component*
Component = *NDC_Polyline*

NDC_Polyline = *NDC_Points*
NDC_Points = **list of** *NDC_Point*
NDC_Point = **R** × **R**

DC_Picture = **list of** *DC_Polyline*

DC_Polyline = *DC_Points*
DC_Points = **list of** *DC_Point*
DC_Point = **R** × **R**

States of the system are described by objects of type *GKS*, which is defined to be an ordered pair with first component of type *NDC_Picture* and second component of type *DC_Picture.*

The *NDC_Picture* is modelled as a list of objects of type *Component*, which, in this case, is simply of type *NDC_Polyline*: a list of real co-ordinate pairs (the co-ordinates of its vertices). As the development proceeds, the need for the introduction of the type *Component* here will become apparent.

The DC picture that is displayed on the workstation is modelled similarly to the NDC picture, as a list of DC polylines. At the DC level, a polyline is also described as a list of real co-ordinate pairs. The type *DC_Polyline* captures the essential features of a polyline displayed on a workstation.

The use of the data type **list** rather than **set** in the definitions of both the NDC and DC pictures allows the order in which primitives are created to be retained for use in their display. The GKS document does not actually prescribe an order of display of primitives, but most implementations do preserve order of creation in order of display.

The definition of the *add_polyline* operation to model the creation and display of polyline output primitives is now given.

The operation

> **let** $mk\_gks(ndcp, dcp) = gks$ **in**
>
> $add\_polyline: GKS \times NDC\_Points \rightarrow GKS$
> $add\_polyline(gks, pts, gks') \triangleq$
> **post** $ndcp' = mk\_ndc\_polyline(pts) :: ndcp \wedge$
> $\quad dcp' = mk\_dc\_polyline(t(pts)) :: dcp$
>
> $t: NDC\_Points \rightarrow DC\_Points$

The **let** clause preceeding the operation definition holds over all subsequent operation definitions and serves to name the initial state and its components. The names of the final state and its components which result from the operation are obtained by decorating the names in this **let** clause with a prime ($'$). A second **let** clause which does exactly this has been omitted for simplicity. The convention followed is that type names have capitalised initial letters and the names of instances are the lower case equivalents of their types.

The first line of the operation definition is its signature, which defines types of the arguments and the result of the operation. The second line simply names the objects of the types given in the signature. Thus *gks*, the initial state, is of type *GKS*, the argument *pts* is of type *NDC_Points* (a list of NDC co-ordinate pairs), and the result of the operation, *gks*$'$, the final state, is also of type *GKS*.

The precondition has been omitted as it is simply **true**, i.e. all values of inputs and initial state will produce a valid result. The postcondition describes the effect of the operation. It states that as a polyline is created, an object of type *NDC_Polyline* is formed (by *mk_ndc_polyline*(*pts*)) and is added to the list, *ndcp*, representing the NDC picture by the *cons* ('::') operator. A corresponding polyline is displayed in DC space by transforming the list of NDC co-ordinates to a list of DC co-ordinates by means of the function *t* and then forming an object of type *DC_Polyline*, which is added similarly to the DC picture. The details of the function *t* are not of interest here, and so a definition of *t* is not given.

That concludes the first specification.

4 The second specification

4.1 GKS segment storage

The next concept to be introduced is one which allows an application some means of structuring a picture. The kind of structuring often required is the ability to define a graphical object, for example, a tree by a sequence of polyline primitives, and the ability to reuse this definition. GKS allows output primitives to be grouped together into units termed segments which are stored, conceptually at the NDC level, and which may be manipulated in certain ways as a single entity. Segments may not be nested.

So pictures are now constructed from both segments and primitives outside segments. The specification will concern itself with capturing this picture structure and the creation and storage of segments, but not with any further segment manipulations.

This enhanced system is now specified.

4.2 The specification with segments

The concepts that now need to be modelled in the state are: the picture at NDC level, the segment store and the picture at DC level. The polyline function is captured as before by an *add_polyline* operation and a new operation, *add_segment*, is required to capture the GKS functions concerned with collecting primitives into segments.

The *add_segment* operation may be considered to be an abstraction of the following sequence of GKS functions:

```
CREATE SEGMENT(· · ·)
POLYLINE(· · ·)
. . .
POLYLINE(· · ·)
. . .
CLOSE SEGMENT
```

The new state and operation definitions are now given.

The state

GKS = *NDC_Picture* × *DC_Picture* × *Segment_Store*

NDC_Picture = **list of** *Component*
Component = *NDC_Polyline*|*Segment*

Segment = **list of** *NDC_Polyline*

Segment_Store = **list of** *Segment*

DC_Picture = **list of** *DC_Polyline*

The state *GKS* has now become a triple with the addition of a third component to model the segment store.

The NDC picture is still modelled as a list of *Component* but now a *Component* may be either of type *NDC_Polyline*, as before, or of type *Segment* ('|' denotes alternation). This allows pictures in NDC space to be constructed from both primitives outside segments and segments.

Segments are modelled, as might be expected, by lists of primitives and segment store is modelled as a list of segments.

The definitions of the types comprising *NDC_Polyline* and *DC_Polyline* have not changed from the previous specification and so have not been repeated. Notice that although the definition of the NDC picture has changed, that of the DC picture has not. Segment storage is only operative in the NDC picture; the picture displayed on the workstation has, conceptually, no segment structure.

The definitions of the operations over this new state require a new **let** clause which also names the additional component of the state which models the segment store. Apart from the fact that it operates over the new state, the definition of the *add_polyline* operation remains unchanged and so has not been repeated here. So, only the definition of the new operation *add_segment* needs to be given. The function t also remains as it was before.

The operations

```
let $mk\_gks(ndcp, dcp, ss) = gks$ in

$add\_segment: GKS \times Segment \rightarrow GKS$
$add\_segment(gks, s, gks') \triangleq$
post $ndcp' = s :: ndcp \wedge$
     $ss' = s :: ss \wedge$
     $dcp' = to\_dc(s) \,||\, dcp$

$to\_dc: Segment \rightarrow DC\_Picture$
$to\_dc(s) \triangleq$ if $s = < >$ then $< >$
          else let $mk\_ndc\_polyline(pts) =$ hd $s$ in
                   $mk\_dc\_polyline(t(pts)) :: to\_dc($tl $s)$
```

The function *to_dc* generates the DC picture corresponding to the segment given as argument. It constructs a list of DC polylines by traversing the list of NDC polylines comprising the segment and generating the corresponding DC polylines by transforming the co-ordinates to DC co-ordinates.

The postcondition of *add_segment* states that the segment *s* is simply added to the lists representing the NDC picture and the segment store. A flattening and loss of the segment structure occurs in adding the DC equivalent of the segment to the DC picture as the list of DC polylines resulting from the application of the *to_dc* function is appended by the operator '||' to the previously displayed picture.

That concludes the second specification.

5 The third specification

5.1 Appearance of primitives

So far, nothing has been said about the appearance (line style, colour etc.) of primitives displayed on workstations.

In GKS the appearance of a primitive displayed on a workstation is determined by its parameters and additional data termed **aspects.** The aspects of a polyline are: linetype, which in GKS may be solid, broken, dashed-dotted or implementation-dependent; linewidth scale factor, which is applied to the nominal linewidth provided by the workstation to give a value which is then mapped to the nearest available linewidth; and polyline colour index.

In the following specifications, linewidth scale factor will be simplified to be just linewidth and it will be assumed that the workstation supports any linetype or linewidth values requested as the GKS mechanism for mapping requested values onto available values will not be specified. Only the two aspects of linetype and linewidth will be looked at, and colour will not be considered.

There are two basic schemes in GKS for specifying aspects, termed **individual specification** and **bundled specification.** In this paper only bundled specification is considered. Individual specification and the relation between bundled and individual specification is addressed in Reference 8.

In the bundled mode of specifying polyline aspects, the values of all the aspects are determined by a single **attribute**, called the **polyline index**. A polyline index defines a position in a table, the **polyline bundle table**, each entry in which is termed a **bundle** and specifies the values for each of the aspects. The bundle corresponding to a particular polyline index is termed the **representation** of the index. There is an operation which sets the value of polyline index modally, as well as an operation to set the representation of a bundle index.

When a polyline is created, the current value of the polyline index is bound to the primitive and cannot subsequently be changed. Bundles are bound to primitives when they are displayed. In GKS each workstation has its own polyline bundle table, which allows the application to control the appearance of polylines created with the same polyline index independently on each workstation on which they are displayed, using the capabilities of the workstation.

5.2 The specification including appearance

The models of the NDC picture, DC picture and segment store are unchanged except for the descriptions of polylines in NDC and DC space. The GKS state is extended by an additional component to model the polyline bundle table. The concept of a workstation in this system is now being represented by two state components: the DC picture and the polyline bundle table.

The operations over this state are again *add_polyline* and *add_segment*, and there is also a new operation, *set_polyline_representation*, which associates a bundle representation with a polyline index in the polyline bundle table.

Both the *add_polyline* and *add_segment* operations are abstractions of an amalgamation of GKS functions. In GKS, polyline index is set modally, but in the specification given here the appropriate polyline indices are provided as arguments to the *add_polyline* and *add_segment* operations (in the latter case bound to the polylines in the list of NDC polylines given as argument). Thus *add_polyline* is equivalent to the sequence of GKS functions:

```
SET POLYLINE INDEX(···)
POLYLINE(···)
```

and *add_segment* is equivalent to the sequence of GKS functions:

```
CREATE SEGMENT(···)
SET POLYLINE INDEX(···)
POLYLINE(···)
...
SET POLYLINE INDEX(···)
POLYLINE(···)
...
CLOSE SEGMENT
```

The new definitions of the state and the operations are given below.

The state *GKS* is now a tuple consisting of four components. The structure of the NDC and DC pictures and of the segment store is unchanged. However, the definition of *NDC_Polyline* has been enhanced by the addition of *Polyline_Index*, which will be bound to the polyline at the time of its creation. *DC_Polyline* has been enhanced by the addition of *Bundle*, which captures the concept of a bundle being bound to the primitive at display time.

The data type used to model the polyline bundle table is a mapping from a polyline index to a bundle (a linetype, linewidth pair). A mapping is a finite function in which the pairing of domain and range elements is constructed explicitly.

The state

```
GKS = NDC_Picture × DC_Picture × Segment_Store
                              × Polyline_Bundle_Table

NDC_Picture = list of Component
Component = NDC_Polyline | Segment
Segment = list of NDC_Polyline

NDC_Polyline = NDC_Points × Polyline_Index
Polyline_Index = N
Segment_Store = list of Segment

DC_Picture = list of DC_Polyline
DC_Polyline = DC_Points × Bundle

Bundle = Linetype × Linewidth
Linetype = N
Linewidth = R

Polyline_Bundle_Table = map Polyline_Index to Bundle
```

The definitions of the operations are given below.

The precondition of *add_polyline* demands that a representation has been defined for the polyline index supplied as an argument. In the *add_polyline* operation, an NDC polyline is now formed from the NDC co-ordinate list and polyline index supplied as arguments; thus the polyline index is bound to the polyline at the time of its creation. The corresponding DC polyline is formed from the transformed NDC co-ordinates and the bundle obtained by applying the polyline bundle table mapping to the polyline index (denoted by *pbt*(*pi*)). This achieves the effect of binding the bundle to the polyline at the time of its display.

The *add_segment* operation has been extended by the addition of a precondition stating that all the polyline indices in the segment supplied as an argument must be defined. The function *to_dc* again traverses a segment, generating the corresponding list of DC polylines by not only transforming the co-ordinates of the vertices, but also by binding in the representation of each polyline index obtained from the polyline bundle table.

The *set_polyline_representation* operation describes the addition to the polyline bundle table of the new representation specified for the polyline

```
let $mk\_gks(ndcp, dcp, ss, pbt) = gks$ in

$add\_polyline: GKS \times NDC\_Points \times Polyline\_Index \rightarrow GKS$
$add\_polyline(gks, pts, pi, gks') \triangleq$
pre $pi \in$ dom $pbt$
post $ndcp' = mk\_ndc\_polyline(pts, pi) :: ndcp \wedge$
     $dcp' = mk\_dc\_polyline(t(pts), pbt(pi)) :: dcp$

$t: NDC\_Points \rightarrow DC\_Points$

$add\_segment: GKS \times Segment \rightarrow GKS$
$add\_segment(gks, s, gks') \triangleq$
pre $\forall pi$ s.t. $mk\_ndc\_polyline(pts, pi) \in elems\ s\ .\ pi \in$ dom $pbt$
post $ndcp' = s :: ndcp \wedge$
     $ss' = s :: ss \wedge$
     $dcp' = to\_dc(s, pbt) \| dcp$

$to\_dc: Segment \times Polyline\_Bundle\_Table \rightarrow DC\_Picture$
$to\_dc(s, pbt) \triangleq$ if $s = < >$ then $< >$
                     else let $mk\_ndc\_polyline(pts, pi) =$ hd $s$ in
                       $mk\_dc\_polyline(t(pts), pbt(pi)) :: to\_dc($tl $s, pbt)$

$set\_polyline\_representation: GKS \times Polyline\_Index \times Linetype$
                     $\times Linewidth \rightarrow GKS$
$set\_polyline\_representation(gks, pi, lt, lw, gks') \triangleq$
post $pbt' = pbt + [pi \rightarrow mk\_bundle(lt, lw)]$
```

index. The operator '+' adds *polyline_index* → *mk_bundle*(*linetype*, *linewidth*) to the mapping, overriding any previous value associated with *polyline_index*.

That concludes the third specification.

6 The fourth specification

6.1 Dynamic behaviour of aspects

It has been noted above that once a polyline index value has been bound to a primitive, the value bound cannot be changed. In the previous specification it was seen that the representation of a polyline index (i.e. the bundle associated with the index in the polyline bundle table) can be changed. From the post-condition of *set_polyline_representation* given in Section 5.2, it is seen that this operation leaves both the DC and NDC pictures unchanged. In other words, changing a representation of a polyline index does not affect the appearance of primitives already created with that index.

GKS does allow changing a representation to have a retrospective effect. If a representation of a polyline index is changed, the appearance of polylines already created with that polyline index may also be changed to the new representation. GKS admits that some workstations may be able to perform such changes dynamically (for example changing a colour table) while others may need to redraw the picture to effect the changes as in the case of a pen plotter. GKS allows the application to control when such redrawing (regeneration) takes place. As the only stored representation of the picture in GKS is that stored in the segment store, such regeneration is performed using the contents of the segment store.

For each picture change that can potentially require a regeneration, each workstation has a flag to indicate whether the change may be performed immediately (IMM) or requires a regeneration (IRG). When a regeneration is required, another flag – **implicit regeneration mode** – is consulted. This flag can be set by the application and has two possible values:

ALLOWED	–	regeneration will be performed immediately
SUPPRESSED	–	regenerations will be postponed until one of the functions REDRAW ALL SEGMENTS ON WORKSTATION, UPDATE WORKSTATION or CLOSE WORKSTATION is invoked.

The system described below is rather more rigidly defined than GKS prescribes, in that here a regeneration is performed in all circumstances where the GKS document does not state anything stronger than that it is allowed.

6.2 The full specification

The state is once again extended, this time by the addition of two flags to control the effects of polyline bundle representation changes, and when any regeneration occurs. The workstation concept is now being captured by four components of the state representing: the DC picture, the polyline bundle table, the bundle modification flag and the implicit regeneration mode.

One new operation, *redraw_all_segments* is defined to represent all the GKS functions which update the picture displayed on the workstation. The definitions of *add_polyline* and *add_segment* remain unchanged, but that of *set_polyline_representation* must now capture the effects of changing a polyline bundle representation on polylines already displayed on the workstation.

The complete state and operation definitions are given below.

The state

$$
\begin{aligned}
&\mathit{GKS} = \mathit{NDC\_Picture} \times \mathit{DC\_Picture} \times \mathit{Segment\_Store}\\
&\qquad \times \mathit{Polyline\_Bundle\_Table}\\
&\qquad \times \mathit{Bundle\_Modification\_Flag} \times \mathit{Implicit\_Regeneration}\\
\\
&\mathit{NDC\_Picture} = \textbf{list of } \mathit{Component}\\
&\mathit{Component} = \mathit{NDC\_Polyline} \mid \mathit{Segment}\\
&\mathit{Segment} = \textbf{list of } \mathit{NDC\_Polyline}\\
\\
&\mathit{NDC\_Polyline} = \mathit{NDC\_Points} \times \mathit{Polyline\_Index}\\
&\mathit{NDC\_Points} = \textbf{list of } \mathit{NDC\_Point}\\
&\mathit{NDC\_Point} = \mathbf{R} \times \mathbf{R}\\
\\
&\mathit{Polyline\_Index} = \mathbf{N}\\
\\
&\mathit{Segment\_Store} = \textbf{list of } \mathit{Segment}\\
\\
&\mathit{DC\_Picture} = \textbf{list of } \mathit{DC\_Polyline}\\
&\mathit{DC\_Polyline} = \mathit{DC\_Points} \times \mathit{Bundle} \times \mathit{Polyline\_Index}\\
\\
&\mathit{DC\_Points} = \textbf{list of } \mathit{DC\_Point}\\
&\mathit{DC\_Point} = \mathbf{R} \times \mathbf{R}\\
\\
&\mathit{Bundle} = \mathit{Linetype} \times \mathit{Linewidth}\\
&\mathit{Linetype} = \mathbf{N}\\
&\mathit{Linewidth} = \mathbf{R}\\
\\
&\mathit{Polyline\_Bundle\_Table} = \textbf{map } \mathit{Polyline\_Index} \textbf{ to } \mathit{Bundle}\\
\\
&\mathit{Bundle\_Modification\_Flag} = \{\mathit{IRG}, \mathit{IMM}\}\\
&\mathit{Implicit\_Regeneration} = \{\mathit{ALLOWED}, \mathit{SUPPRESSED}\}
\end{aligned}
$$

The only changes to the previous state are the addition of the two flags: bundle modification flag and implicit regeneration mode. The definition of *DC_Polyline* has also been extended to include the polyline index for reasons that will become clear in the next section.

The function *regenerate* traverses the list representing the segment store, using the function *to_dc* to generate from the NDC polylines in each segment the corresponding DC polylines, with both polyline indices and bundles bound. The function *recreate* describes the effect of an immediate change to a DC picture by effectively rebinding the bundles to the DC polylines in the DC picture. It uses the polyline index value now contained in each DC polyline to look up the new bundle representation in the polyline bundle table mapping. This is done without reference to the segment store.

```
let mk_gks(ndcp, dcp, ss, pbt, bmf, ir) = gks in

add_polyline: GKS × NDC_Points × Polyline_Index → GKS
add_polyline(gks, pts, pi, gks′) ≜
pre pi ∈ dom pbt
post ndcp′ = mk_ndc_polyline(pts, pi) :: ndcp ∧
     dcp′ = mk_dc_polyline(t(pts), pbt(pi), pi) :: dcp

t: NDC_Points → DC_Points

add_segment: GKS × Segment → GKS
add_segment(gks, s, gks′) ≜
pre ∀ pi s.t. mk_ndc_polyline(pts, pi) ∈ elems s . pi ∈ dom pbt
post ndcp′ = s :: ndcp ∧ ss′ = s :: ss ∧ dcp′ = to_dc(s, pbt) || dcp

to_dc: Segment × Polyline_Bundle_Table → DC_Picture
to_dc(s, pbt) ≜ if s = < > then < >
                else let mk_ndc_polyline(pts, pi) = hd s in
                     mk_dc_polyline(t(pts), pbt(pi), pi) :: to_dc(tl s, pbt)

redraw_all_segments: GKS → GKS
redraw_all_segments(gks, gks′) ≜
post dcp′ = regenerate(ss, pbt)

regenerate: Segment_Store × Polyline_Bundle_Table → DC_Picture
regenerate(ss, pbt) ≜ if ss = < > then < >
                      else to_dc(hd ss, pbt) || regenerate(tl ss, pbt)

set_polyline_representation: GKS × Polyline_Index
                             × Linetype × Linewidth → GKS
set_polyline_representation(gks, pi, lt, lw, gks′) ≜
post pbt′ = pbt + [pi → mk_bundle(lt, lw)] ∧
     (bmf = IMM ⇒ dcp′ = recreate(dcp, pbt′)) ∧
     (bmf = IRG ∧ ir = ALLOWED ⇒ dcp′ = regenerate(ss, pbt′)) ∧
     (bmf = IRG ∧ ir = SUPPRESSED ⇒ dcp′ = dcp)

recreate: DC_Picture × Polyline_Bundle_Table → DC_Picture
recreate(dcp, pbt) ≜ if dcp = < > then < >
                     else let mk_dc_polyline(dpts, b, pi) = hd dcp in
                          mk_dc_polyline(dpts, pbt(pi), pi)
                          :: recreate(tl dcp, pbt)
```

6.3 The effect of a regeneration

As a simple illustration of the ability of formal specification to explain the consequences of design decisions, consider the following.

If an implicit regeneration is performed, the postcondition of *set_polyline_representation* says that:

$$dc\_picture' = regenerate(segment\_store,\ polyline\_bundle\_table')$$

From the postconditions of *add_polyline* and *add_segment* it is seen that only *add_segment* stores primitives in segment store. Thus the effect of a regeneration is to remove all primitives outside segments from the picture displayed on the workstation in the process of applying the new polyline bundle table to change the representations of displayed polylines. This is indeed what happens in GKS.

7 Conclusions

This paper has shown how the specification of a complex system can be built up in stages and how the resulting specification can be used to deduce behavioural properties from the system. This latter point is further developed in References 6 and 8.

References

1 HILL, I.D. and MEEK, B.L.: *'Programming language standardisation'*, Ellis & Horwood, 1980.
2 HOPGOOD, F.R.A., DUCE, D.A., GALLOP, J.R. and SUTCLIFFE, D.C.: *'Introduction to the Graphical Kernel System (GKS)*, Academic Press, 1983.
3 'Information processing systems — Computer graphics — Graphical Kernel System (GKS) functional description', ISO 7942, ISO Central Secretariat, Geneva, 1985.
4 TURNER K.J.: 'Towards better specifications', *ICL Tech. J.*, 1984, **4** (1), 33–49.
5 SUFRIN, B.: 'Towards a formal specification of the ICL Data Dictionary', *ICL Tech. J.*, 1984, **4** (2), 195–217.
6 DUCE, D.A., FIELDING, E.V.C. and MARSHALL, L.S.: 'Formal specification and graphics software', RAL-84-068, Rutherford Appleton Laboratory, 1984.
7 JONES, C.B.: *'Software development: a rigorous approach'*, Prentice-Hall, Englewood Cliffs, New Jersey, 1980.
8 DUCE D.A. and FIELDING, E.V.C.: 'Better understanding through formal specification', RAL-84-128, Rutherford Appleton Laboratory, 1984.

The effects of inspections on software quality and productivity

B.A. Kitchenham
ICL Software Engineering Technical Centre
Kidsgrove, Staffordshire
A.P. Kitchenham
ICL Mainframe Systems Division
Kidsgrove, Staffordshire
J.P. Fellows
Industrial student, North Staffordshire Polytechnic
Blackheath Lane, Stafford

Abstract

The paper reports the effect of using inspections during the early phases of software production on two developments which were part of the ICL VME operating system. The results indicated that detailed design inspections improved the quality of code, both by removing design errors and by the fact that errors introduced subsequent to design were found at earlier stages in the software testing phase.

Design inspections involved only a small proportion of the total development effort (6–9%) but accounted for a substantial proportion (40–50%) of all the recorded errors. The cost of finding errors in terms of man-hours was also small (1·2 to 1·6) compared to an observed cost of 8·47 man-hours per error for errors found by code execution in one of the developments.

1 Introduction

Since 1981, the production of ICL's VME Operating System has been instrumented by a semi-automated data collection and analysis scheme[1]. One of the aims of this scheme was to develop the means to evaluate the effectiveness of VME production methods. A previous paper[2] has described how the initial analyses indicated that VME error screening methods were allowing a relatively high proportion of design errors compared with code errors to reach our in-house services and our customers. In addition results indicated a relationship between the type of error found and the method of error screening used, with some evidence that non-execution techniques were important for screening design errors. The conclusion of that work was that Inspection methods[3], which are non-execution methods and apply equally to the design and coding phases of software production, would potentially address the problems identified.

The advantage of the data collection and analysis scheme is that it may be used to evaluate the effect of change as well as indicate potential problems. This implies that the introduction of Inspections into VME production methods can and should be evaluated. This paper, therefore, reports the effect of using Inspection methods, by examining two VME developments on which the techniques were used.

2 Inspections

Although it was intended to use the method of Formal Inspection developed by Michael Fagan of IBM[3], there were a number of difficulties due to a lack of proper training in the technique and a number of compromises were made.

The original idea behind Inspections was that of applying process management techniques to software development. Fagan pointed out that successful management requires planning, measurement and control, and that software management would be improved by incorporating these elements into a defined development process comprising a series of operations with associated exit criteria. Inspections, themselves, were intended to allow the completeness and correctness of the software product to be assessed during the early stages of development, and thus to form part of a process of determining whether exit criteria have been achieved.

The Inspection process involves a group of people checking some product documents (e.g. design documents, code listings, test plans, etc.) at specific points in the development process in order to find any errors efficiently and economically.

Fagan recommends an inspection team take on the following roles:

Moderator	who manages the inspection process. He or she is responsible for scheduling the inspection meeting, circulating inspection material, chairing the inspection meeting, reporting inspection results and following up any identified rework. A moderator must be a competent software engineer but need not be a technical expert on the software being inspected.
Designer	the person responsible for producing the design of the software.
Coder/implementor	the person responsible for translating the design into code.
Tester	the person responsible for writing and/or executing test cases or otherwise testing the product of the designer and coder.

If the coder is also the designer and tester, he or she should act as the designer and other implementors should be asked to act as coder and tester.

The processes involved in Inspections are:

- Overview (for design inspections, not code inspections)
- Preparation
- Inspection
- Rework
- Follow-up.

Each of these processes will now be described in more detail.

Overview

The overall area being addressed by the software and the specific area currently due for Inspection are described by the designer. Design documentation should be circulated to participants after the overview.

Preparation

Each person should study the inspection material on their own in order to understand its logic and intent. Fagan recommends that inspection teams should study the ranked distribution of error types found by recent inspections, and checklists of error detection guidelines.

Inspection

A ‘reader’ is chosen by the moderator to describe how the next phase of development will progress given the current phase. Thus, at a design inspection, the coder will describe how the design will be implemented, and will paraphrase the design, covering each piece of logic and each branch at least once.

The objective of the Inspection is to *find errors* not to provide solutions. Once an error is recognised it is noted by the moderator, its type is classified and its severity identified. Within one day of the inspection, the moderator should produce a written report of the inspection and its findings.

Rework

All errors or problems noted in the inspection report need to be resolved by the designer or coder/implementor.

Follow-up

It is the responsibility of the moderator to ensure all errors and problems are resolved and to schedule re-inspections if necessary.

There were three main differences in the VME study compared with Fagan's procedures:

(i) the number of people present at each Inspection varied: for high level design inspections (i.e. inspections of the overall subsystem design and interface modules), more people attended than the recommended number; for detailed design inspections (i.e. inspections of the internal design of a module) fewer people attended than recommended; and code inspections were often a dry checking exercise (i.e. code-reading exercise) performed by an individual other than the coder,
(ii) the moderator and other participants were people involved in the development itself,
(iii) there were no suitable checklists available.

The procedures that were adopted were:

(i) for all design inspections a moderator was present to control the inspection and record all the identified problems,
(ii) the inspections were directed to identifying problems not solving them,
(iii) the inspections were kept to approximately two hours elapsed per session,
(iv) each problem was assigned to an actionee to solve and the moderator was responsible for ensuring that a solution was found,
(v) inspection reports were normally kept and copies sent to project managers,
(vi) errors were classified for later analysis.

3 Development 1

3.1 Background

This development was concerned with a new tape management facility. It was a moderately complex development. The staff involved consisted of a team leader/designer who was very experienced in tape management problems and a number of other implementors and testers who were not experienced in such problems although they were all experienced in VME development.

3.2 The design inspection process

The development was subject to one high level design inspection which found 30 problems. The 73 constituent programs making up the development were identified and some of those programs were then given detailed design inspections.

The programs not given detailed design inspections were those that the team leader believed were particularly simple or well understood. There were 30 such programs.

The remaining 43 programs were all given detailed design inspections. This took a total of 9 sessions and found 213 problems. The time attributed to inspection sessions was 95 hours. Allowing two hours preparation time for every hour spent in inspections would increase the time to 285 hours which means that each error took an estimated 1·17 man hours to identify or the error screening rate was 0·85 errors per hour. In addition since the total time devoted to the development was 4830 hours, the proportion of the total development effort devoted to inspections was only 6%.

3.3 Evaluation of the inspection process

The fact that this development was split into two groups of programs, one group given detailed design inspections and one group not given detailed design inspections allows the effect of the inspections to be gauged by looking at the subsequent error history of the two groups. Table 1 shows this in terms of a breakdown of the error rate at various stages of the development process.

It is clear from Table 1 that the team leaders' assessment of the programs not given detailed design inspections was correct. The lower overall error rate indicates that they were indeed simpler. However, it is interesting to note that although the programs given detailed design inspections did have a larger proportion of subsequent errors, the errors were found earlier in the

Table 1 The error rates for development 1 subsequent to the design phase

	Programs given detailed design inspections	Programs not given detailed design inspection
Number of programs	43	30
Size of programs (lines of code)	8653	4681
Subsequent error rate per 100 lines:		
Due to dry checking	0·99	0·53
Due to code execution testing	0·80	0·68
Due to use on in-house services	0·20	0·28
Found by customers	0·01	0·04
Overall	2·00	1·54

subsequent software development cycle. Thus, by the time the software had been in the field 9 months, the programs given detailed design inspection revealed 1 error in 8653 lines of code compared with the programs not given detailed design inspections which revealed 2 errors in 4681 lines. The difference in the subsequent distribution of errors is statistically significant ($p < 0.05$) using a chi-squared test.

4 Development 2

4.1 Background

This development was to provide an automatic filestore management system for backups and archiving. This was a complex development. The staff involved were all experienced in VME design and implementation but had limited experience in the problem area.

4.2 The design inspection process

In order to complete the high level design inspection of this development, 15 inspection sessions were required which found 197 errors. Not all detailed design inspection sessions were properly recorded but for those for which records remained, 21 sessions found 307 errors.

The total time attributed to these inspection sessions was 265 man hours. Allowing 2 hours preparation time for every hour spent in inspections would increase the time to 795 man hours. This would imply that each error took 1·58 man hours to find, or the error screening rate was 0·633 errors per hour.

Detailed design inspection records were not available for 31% of the subsequent code. On a pro-rata basis this implies that an additional 117 man hours were spent on such inspections. The total time devoted to this development was 10 164 man hours which implies that design inspections (detailed and high level) accounted for approximately 9% of the development time.

4.3 Evaluation of the inspection process

Since this development did not have any internal comparisons, the only method of evaluating the effect of the inspections is to compare the development with overall trends.

In terms of overall productivity the development involved the production of 39 000 lines of code at an overall rate of 144 lines per man week in an elapsed time of 24·5 months. Compared with the Rome Air Development Centre statistics for US developments quoted by Putnam[9], the productivity achieved for this development was 30% better than average although the time scale was slightly longer than average.

Table 2 Distribution of post-design errors

Type of error	% Errors found in development 2 subsequent to design	% Errors found in all VME developments between Jan 1983 and June 1983
Dry checking and/or code inspection	57·5	37·6
Dry execution testing	38·4	51·2
Use on in-house services	4·1	11·2

In terms of subsequent error occurrence, the development can be compared with the overall average for VME developments during a comparable time period as shown in Table 2. This indicates that the post-design errors are being found earlier in the subsequent development cycle compared with VME overall. This confirms the trend observed for development 1.

Table 3 shows the type of error found after design. The difference between the percentages of errors classified as 'other' is probably due to the fact that these include source clearance errors and ripple errors which are more likely to occur in old code (as much of VME is) than in newly developed code and development 2 involved the development of an entirely new subsystem. The differences of particular interest are the decrease in design errors and the increase in interface errors compared with overall VME trends. The decrease in design errors is hopefully attributable to the design inspections. However, the increase in interface errors needs further explanation and will be discussed later.

The relationship between program size and number of post-design errors is shown in Fig. 1. This shows a fairly linear relationship which accounts for a statistically significant ($p < 0{\cdot}001$) 54% of the variation. The existence of a linear-type relationship has been observed for VME developments previously[5] and indicates that the design inspection process does not radically change the nature of post-design implementation. The existence of the relationship allows programs with particularly high or particularly low error rates to be identified. Using an *ad hoc* procedure the diagram shown in Fig. 2 was used to separate the programs into 3 groups. The group of programs

Table 3 Type of post-design errors

Type of error	% Errors found in development 2	% Errors found in all VME development
Software interface	5·8	2·1
Design error	8·1	13·3
Code error	81·0	70·3
Other	5·1	14·2

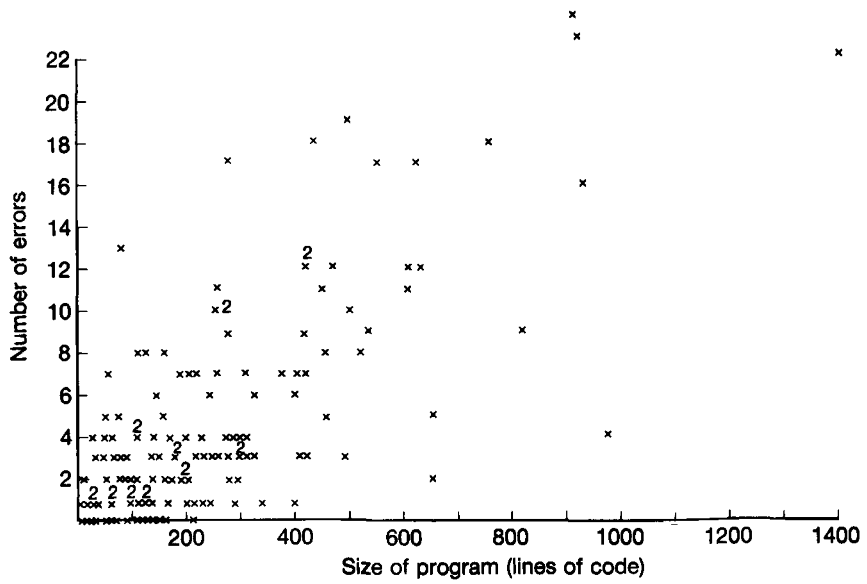


Fig. 1 Scatter diagram of the number of errors against program size for development 2

size	0	1	2	3	4	5	6	7	8	9	10	11	12	13–18	19+	Total
0–50	16	5	1	1	2											25
51–100	6	4	4	4	1	2		1						1		23
101–150	7	5	2	3	3		1		2							23
151–200	1	1	4	2	1	1		1	1							12
210–300	1	5	3	5	5		1	3		1	2	1		1		28
301–400		1		4	1		1	2								9
401–500		1		3		1	1	2	1	1	1	1	3	1	1	17
501–600									1	1				1		3
601–700			1			1						1	2	1		6
701–800														1		1
801–900										1					1	2
901–1000					1									1	1	3
Rest															1	1
Total	31	22	15	22	14	5	4	9	5	4	3	3	5	7	4	

Fig. 2 Classification of programs based on the relationships between program size and number of errors for development 2

below the thick semi-diagonal lines was that which had a high error rate. This division identifies a group of 20 high error rate programs which accounted for 9·6% of the code and 26% of the post-design errors, and a group of 212 low error rate programs which accounted for 18·4% of the code and only 4% of the post-design errors.

The programs which revealed a high error rate were primarily those that interface with another VME subsystem developed by another production team which played an essential support role for development 2. These programs were responsible for the large number of interface problems noted earlier. The reason why these problems were not revealed during the design process is probably due to the fact that no representative from the support subsystem attended the inspections. This emphasises the need for all affected areas to be represented in inspections and indicates why for high level inspections the experience within VME has been for more people than needed in Fagan's recommended number.

Finally, the cost effectiveness of inspections may be considered by comparing the cost of finding an error during the design inspections (1·58 man hours) with the cost of funding an error during execution testing and in-house use. Test and support effort amounted to 2854·6 man hours and execution testing and in-house use revealed a total of 337 errors. Thus the cost of finding errors later in the development cycle is approximately 8·47 man hours which is more than 5 times more costly.

5 Comparison with other results

Fagan[6] has stated that within IBM, Inspections find 80% of errors that are observed prior to system release. For VME the results are a bit more difficult to gauge because the VME code inspections were much more like dry checking than inspections. However, considering all non-execution techniques together (design inspections plus code inspection and dry checking), these accounted for 73% and 75% of the recorded errors of development 1 and development 2 respectively while the design inspections alone accounted for 50% and 41% respectively of the recorded errors of each development.

The rates of the high level and detailed design inspections are shown in Table 4. These are much faster than the rates recommended by Fagan[6] which are 300 and 135 lines per elapsed hour for what Fagan calls a design 'overview' and design inspection.

Table 4 Rate of design inspections in equivalent lines of code per elapsed hour

	High level design	Detailed design
Development 1	520	480
Development 2	650	640

Thus, although the rate of the design Inspections was too fast by IBM standards, if the results of dry checking and code inspections are included, the error screening rate of non-execution methods is close to IBM standards (i.e. code execution techniques which logically reveal code errors efficiently).

6 Discussion

The results described in this paper both support the contention that Inspections are an efficient and inexpensive method of error screening, and confirm the diagnosis presented in a previous paper[2] of some of the problems associated with VME code production and how they might be solved.

The effect of the design inspections seems to be not only to find design errors but also to allow subsequent errors to be found earlier in the development process. If the process of system production is considered in the manner suggested by Small and Faulkner[7], as a process of design decomposition followed by system integration, then errors are introduced into each level in the system decomposition process and unless the errors are removed at each stage they propagate throught the subsequent levels. In these terms, the results of Inspections could be explained as a process of error screening relevant to the particular stage of system decomposition which prevents the propagation of errors to lower stages in the decomposition process. This ensures that the errors introduced subsequently are those which may be found efficiently by subsequent screening techniques (i.e. code execution techniques which logically reveal code errors efficiently).

A complementary model of software errors[8] considers the type of errors found during each stage of system integration and suggests that the type of error found is related to an equivalent stage of decomposition phase. So that code errors are found during unit testing, detailed design errors are found during subsystem testing, high level design errors are found during system testing, and requirements errors are found after release to customers. Boehm[9] reports TRW findings which suggest that it costs between 20 and 50 times more to deal with errors in a released product compared with errors during the requirements and high level design phases of development. Thus, Inspections should provide means of screening the most expensive errors. Obviously it is impossible to assess objectively at which stage an error would have been found subsequently (unless it is left in and tracked) but the subjective feeling of those involved with Inspections was that they were finding errors which would otherwise have reached customers. This has encouraged the introduction of Inspections even earlier in the software development process (i.e. requirements definition).

Acknowledgements

We should like to thank Mike Fagan and Tom Gilb for their encouragement and advice on the subject of Inspections.

References

1 KITCHENHAM, B.A.: 'Program history records – a software data collection and analysis scheme' ICL Tech. J. (1984).
2 KITCHENHAM, B.A.: 'The use of software metrics to assess software development techniques' Proc. FTCS – 13th Annual International Symposium on a Fault-Tolerant Computing, Milan (1983).
3 FAGAN, M.E.: 'Design and code inspections to reduce errors in program development' IBM Sys. J., 15, 3 (1976).
4 PUTNAM, L.H.: Tutorial on Software Cost Estimating and Life-cycle Control: Getting the Software Numbers. IEEE Computer Press, 1980.
5 KITCHENHAM, B.A.: 'Software metrics'. Proc. Convention Informatique, Vol. A, 244–249 (1982).
6 FAGAN, M.E.: Personal communication (1983).
7 SMALL, M. and FAULKNER, T.: 'A quality Model of System Design and Integration' FTCS-13, Milan (1983).
8 JONES, C.B.: 'Rigorous Design of Software'. Lecture to the North Staffs Branch of BCS, Keele, (1981).
9 BOEHM, B.W.: Software Engineering Economics. Prentice-Hall, Inc, Englewood Cliffs, N.J. (1981).

Recent developments in image data compression for digital facsimile

M.J.J. Holt and C.S. Xydeas
Department of Electronic and Electrical Engineering
Loughborough University of Technology
Loughborough, Leics., England

Abstract

Digital facsimile is an increasingly important component in office automation. Techniques for the compression of bi-level scanned image data are vital to the efficient storage, retrieval and transmission of office documents. Part I of this paper reviews recent developments and evaluates various existing techniques. The review is by no means exhaustive, but covers what the authors consider to be significant developments in the efficient compression of black and white images for an office automation environment. In Part II, a scheme for the compression of typewritten and printed documents is described. It incorporates a new pattern-matching algorithm which can handle a variety of styles and sizes of text more efficiently than existing methods.

PART I – EXISTING COMPRESSION TECHNIQUES

1 Introduction

Digital facsimile is an increasingly important aspect of office automation. Equipment is widely available which can scan and digitize a black and white document at a resolution of 200 dots per inch (7.7 dots per mm.). At this resolution, an A4-size document generates about 0.5 Mbytes of image data. Compression is therefore of great importance for the efficient storage, retrieval and transmission of office documents.

There are three stages in the digitization of a document. First, a scanner samples the intensity of the document at regularly spaced picture elements (pels). Next, the intensity is quantised to two levels (black and white), thereby reducing the image to a binary matrix where 1 = black and 0 = white. Finally, this data may be compressed by eliminating redundant information in the image. This paper is concerned with techniques for the third stage, image data compression.

2 Exact codes and standards

Most of the compression schemes reported before 1980 share two properties. They are 'exact' codes, in the sense that the original image data can be reconstructed exactly from the compressed code. They also exploit only local redundancy, with the advantage that very little of the image data (typically one or two scan lines) are used at any one time in the generation of the code, and hence the memory requirement is low in both the encoding and decoding equipment. Examples of such schemes are run-length coding[1], block coding[2], line-difference coding[3] and predictive coding[4].

By 1978, the proliferation of such schemes was such that the CCITT study group XIV was set up to achieve compatibility between group 3 facsimile apparatus. In 1980, the group recommended a one-dimensional run-length code, known as Modified Huffman code (MHC), and a two-dimensional line-difference code known as the Modified READ Code (MRC), which forms an optional extension to MHC[5]. Other exact codes, based on two-dimensional predictive coding, achieve slightly higher compression than MRC[6,7], but the latter algorithm is preferred because of its relative simplicity.

Since the recommended codes are intended for transmission over noise-prone public switched telephone networks, they include redundant bits which limit the extent of image corruption in the event of a transmission error. In a later study the CCITT proposed a two-dimensional code for noise-free environments (MRC-II), which is equivalent to MRC but without the redundant bits for error protection[8]. The three CCITT-recommended codes (MHC, MRC and MRC-II) can compress a typical A4 business letter, sampled at 200 pels per inch, by factors of 14, 20 and 28 respectively[5,8].

3 Preprocessing

Many existing schemes improve on the compression of exact codes by introducing a preprocessing step, in which some non-essential information in the image is irreversibly removed. The main purpose of preprocessing is to increase the local redundancy in the image data, without seriously distorting the reconstructed image, enabling an exact code to perform more efficiently.

3.1 Smoothing

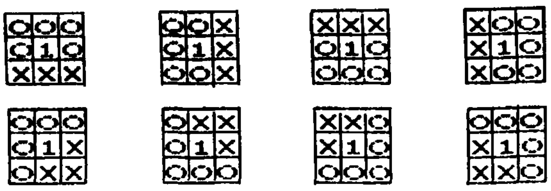
One type of preprocessing involves removing isolated black pels and smoothing the edges of black objects. This can often actually improve the visual quality of the image, by eliminating spurious effects due to noise introduced during digitization.

Smoothing is normally achieved by a set of two-dimensional templates (masks) of a given size, to which the digitized image is forced to conform.

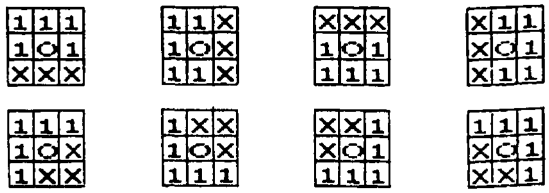
Each mask is laid over the original image data (source) in every possible position, and the value of the key element (normally the central element of the mask) is changed in the image whenever a one-to-one match exists between the rest of the mask and the source. Masks of various sizes are used in reported preprocessing algorithms, typically 3 × 2[9], 3 × 3[10] and 4 × 4[11]. A typical set of 3 × 3 smoothing masks is shown in Fig. 1, and their effect on a sample of image data is shown in Fig. 2.

The improvement in compression as a result of noise removal or smoothing depends very much on the information content and noise content of the image. Improvements between 15% and 50% in compression have been reported as a result of applying preprocessing masks prior to two-dimensional exact coding[9,12].

(a) Masks in which the central pel is changed to white.

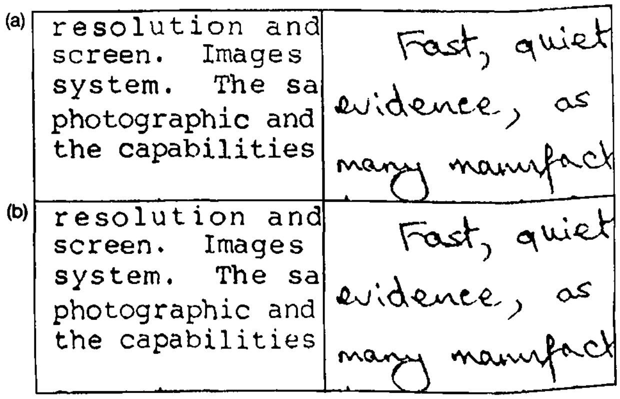


(b) Masks in which the central pel is changed to black.



0 = white 1 = black × = don't care

Fig. 1 Masks used for Majority Logic Smoothing[10]



(a)

resolution and
screen. Images
system. The sa
photographic and
the capabilities

Fast, quiet
evidence, as
many manufact

(b)

resolution and
screen. Images
system. The sa
photographic and
the capabilities

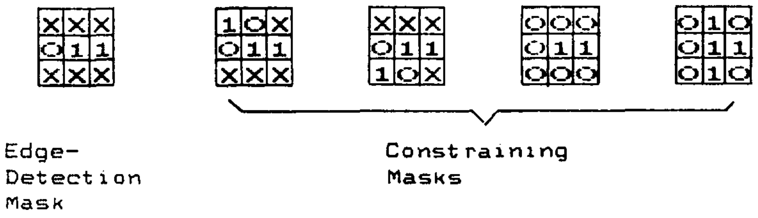
Fast, quiet
evidence, as
many manufact

Fig. 2 Effect on smoothing of samples of typewritten and handwritten documents: (a) Original image. (b) Smoothed image.

3.2 Thinning

A number of existing compression methods[13,14] code only the centre lines, or skeletons, of black objects, achieving compression 50–90% higher than MRC. The image can no longer be reconstructed exactly, but a close approximation is obtainable when the black lines in the original image are of roughly uniform thickness. Many handwritten and hand-sketched images and some technical drawings have this property.

Of the many thinning algorithms reported in the literature, a typical approach used in the context of compression is that described by Judd[13]. Edge pels of black objects are removed by the iterative application of a set of 3 × 3 masks, subject to certain constraints. The algorithm consists of four subcycles, which remove separately the left, right, top and bottom edges of black objects. The masks used in the left-edge subcycle are shown in Fig. 3. If a 3 × 3 neighbourhood matches the edge-detection mask, the central black pel is changed to white, provided that the neighbourhood does not also match any of the three constraining masks. The masks used in the other three subcycles are rotations of the masks in Fig. 3 through multiples of 90 degrees. The constraining masks are included to preserve connectivity in black objects and to prevent the erosion of line-ends.



0 = white 1 = black x = don't care

(central pel is changed to white, masks for other edges are rotations of these through multiples of 90 degrees)

Fig. 3 Thinning masks used by Judd[13], left hand edges only

Unlike smoothing, the effect of thinning is to introduce some distortion in the image. This is usually compensated by a post-processing step at the decoder, which restores the thickness of objects in the reconstructed image close to that of the original. In Judd's scheme[13], for example, black objects are arbitrarily thickened by one pel on the left and top edges. The effects of Judd's thinning and thickening algorithms on a sample of image data are shown in Fig. 4.

4 Shape coding methods

All the methods described so far have used only local redundancy in achieving compression. Another class of methods ('shape' coding) exploits a higher level of redundancy by describing the shapes of objects and lines in an

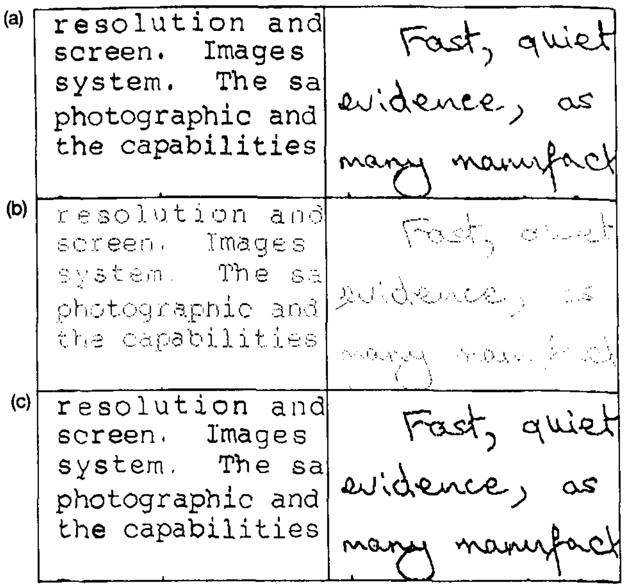


Fig. 4 Effect of thinning on samples of typewritten and handwritten documents: (a) original image, (b) thinned image, (c) thinned image after thickening

image. Such methods are more costly in terms of computation and memory, since they operate on the whole image, or at least a significant part of it, at a time. However, shape coding can achieve high compression for certain types of image, such as handwriting and technical drawings.

4.1 Chain coding

Some coding schemes employ a chain-coding of the edges or centre lines of black objects. Essentially, chain-coding is an efficient way of representing a sequence of adjacent points, in which the position of a point is expressed as a direction relative to the previous point. In an image digitized over a rectangular grid, there are only eight possible directions between adjacent points. Hence three bits per point are required for all but the first point in any chain. A variation of this is differential chain-coding, in which a point position is expressed as a change of direction ('turn') relative to the preceding two points. Since the 'straight-ahead' turn (no change of direction) has a considerably higher probability than other turns, a variable-length coding of the turns can achieve greater compression than straight-forward chain-coding.

Differential chain-coding has been applied to the contours (edges) in a black-and-white image[15]. By reconstructing the contours and filling the spaces between them, the original digitized image can be exactly reproduced from the contour code. However, this method achieves, on average, slightly lower compression than other exact algorithms such as predictive coding[16].

Judd's scheme[13] applies chain-coding to the centre-lines of black objects, which are obtained by thinning. This approach achieves compression typically 50% higher than any exact coding method. However, entropy measurements suggests that a 2-D predictive code applied to the thinned image (in place of chain coding) would generate even higher compression[17]. Furthermore, the approach is only suitable for images where the black lines are of approximately uniform thickness.

4.2 *Vector/arc approximation*

Lines in an image can often be approximated by a sequence of straight lines and circular arcs. This approach is established for preprocessing in syntactic optical character recognition, and for automatic input to CAD systems. It also has potential value in the compression of an image which can be well approximated by vectors and arcs of sufficient length.

One such scheme is described by Pavlidis[18]. The image is first thinned, and lines in each thinned object are tracked and fitted by straight-line segments and circular arcs. Short segments which are unconnected at one end are assumed to be either noise or serifs in characters, and are eliminated. The remaining segments are then encoded as vectors and arcs, which generate fewer bits than exact coding.

The effect of this vector/arc approximation is seen in Fig. 5. There is some visible distortion in the drawing, but the characters are still readable. The technique is clearly unsuited to lines of non-uniform thickness and solid

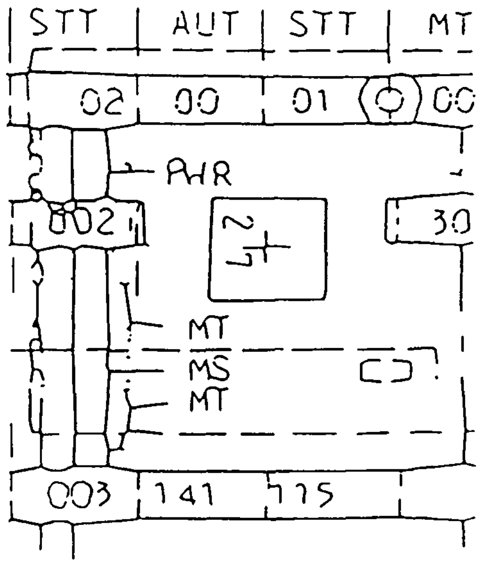


Fig. 5 Effect of thinning and vector/arc approximation on a sample of an engineering drawing. Reproduced from Pavlidis[19]

black areas. No direct comparison has been made between the compression of this and exact-coding methods.
A vector coding which preserves line-width is described by Ramachandran[20]. It has a low memory requirement but still presents the image in a form suitable for analysis and editing. Compression reported for a circuit diagram is lower than exact coding of the same image reported elsewhere[6], but there is still scope for optimization of the vector code[20].

5 Pattern-matching methods

It has been known for some time that for documents which are predominately machine-generated text (typed or printed), much greater compression is achievable by exploiting the redundancy due to the repetition of symbol patterns[21]. Pattern-matching methods for bilevel image data compression fall into two categories:

5.1 Character recognition

A very efficient approach to text image compression uses optical character recognition (OCR). Each character extracted from the document is compared with a pre-programmed library of standard character descriptions, using statistical or structural OCR techniques[22]. If a match is found, the character is represented in the code by its ASCII code (or equivalent). A small amount of additional code is required to convey details of the word, line and paragraph spacing in the document, but the compression factor is extremely high-typically six times that of MRC-II, and greater than 100 for most typewritten business documents. Furthermore, the code is in a form which lends itself to further text processing if required.

Unfortunately, reported implementations of such schemes[23-24] are restricted to a small set of styles and sizes of printed text, or require training specifically to each set of character patterns they are able to handle. They also need to be specifically tailored to any mathematical symbols or foreign alphabets. Existing character-recognition schemes are not therefore well suited to general purpose facsimile coding, where there may be no control over character fonts in the input documents.

5.2 Symbol pattern-matching

A more general-purpose approach is symbol pattern-matching[21], where compression is achieved by encoding only the first occurrence of each distinct symbol pattern in the image. The encoder maintains a copy of the image and a library of symbol patterns in its memory. Initially this library is empty. Each symbol encountered in the image data is extracted and deleted from the memorized image. The symbol pattern is compared with those in the library by template matching, i.e. superimposing two symbols and thresholding the error pattern generated where the patterns differ. If no match is found, the current symbol is added to the symbol library.

For each symbol extracted, its location in the image and its identification are encoded. The identification of a symbol is either its position in the symbol library or an indication that it is to be added to the library. For each new library symbol, a full description must also be encoded, consisting of the size and exact pattern of the symbol. The decoder maintains an adaptive symbol library identical to that constructed by the encoder.

Since the library of symbol descriptions is constructed from the source document itself, symbol pattern-matching is theoretically capable of handling any size or font of character and many different alphabets. A number of practical implementations of this idea have been reported[23,25,26], and the compression factors obtained are two to three times greater than that given by MRC coding. However, the efficiency of such schemes has been found to be considerably less when the text is small or the digitization noise is high[31]. An improvement in this area is presented in the latter half of this paper.

6 Hybrid coding schemes

The shape-coding and pattern-matching methods described above can achieve much greater compression than the more established exact coding algorithms. However, none of these methods is suitable for all types of document images. For an arbitrary document, the best compression can only be obtained by a hybrid scheme which adaptively selects the most suitable of a variety of coding methods.

It can also be beneficial to combine two or more methods in the coding of a single document. A business letter, for example, contains not only typewritten text, for which pattern-matching is ideally suited, but also a company logo and a signature, both of which are better suited to exact coding or shape coding methods. Documents may also contain areas of halftone (grey-scale simulated by black dots), for which specialised coding methods are available[27]. Ideally, an adaptive hybrid scheme should be capable of applying a separate appropriate coding method to each class of imagery in the document.

This is partly achieved by the CSM scheme[23], which combines exact coding and symbol pattern-matching within the same document. Symbols are defined to be connected black objects entirely contained within a fixed-sized moveable window. Any such objects are removed from the image and coded by pattern-matching. The remainder of the image is termed the ‘residue’ and is encoded exactly, using the MRC algorithm. A variation on this approach incorporates in addition character recognition[24]. Symbols are compared first with a preprogrammed library of standard character fonts. Recognised characters are ASCII coded, while unrecognised characters and the residue are handled as in the CSM scheme. This approach works reasonably well for documents which are predominantly machine-generated text, but can be inefficient for other images, since the symbol-extraction algorithm is applied, often needlessly, over the entire image. It also performs poorly over areas of halftone and other forms of shading, where the symbol pattern library can rapidly fill up with many ‘symbols’ which are in fact random patterns of lines or connected dots.

These problems can be overcome by scanning the data prior to processing, measuring basic properties which classify the image into text, graphics, halftone, etc., and performing a field segmention of the image into regions of different classes if necessary. Techniques for segmenting a document into text and graphics have been developed for document reading and analysis systems[28,29]. A fusing process locates contiguous blocks in the image, which are either lines of text or regions of graphics. The classification of each block is determined from its shape and basic run-length statistics.

A method for identifying regions of halftone has been implemented in an experimental facsimile workstation which combines two different exact coding algorithms[30]. The classification is determined, in a prescanning of the document, by evaluation of joint value and slope distributions of 4-bit quantized samples. There is still a need for a scheme which recognises halftone directly from binarized image data.

7 Discussion

As hardware costs have decreased, compression techniques of increasing complexity have become technically feasible. Sophisticated algorithms have been described with the potential for compression several times that achievable by established techniques. However, their practical implementation is not so well documented.

To bring this order of compression into the automatic office, it is necessary to generalise the new techniques to handle arbitrary input documents. This involves further development of the vector and arc approximation approach. Symbol pattern-matching must be extended to handle a wider variety of text sizes. Comprehensive and reliable segmentation algorithms are also required for the realisation of fully adaptive hydrid systems.

PART II – NEW PATTERN-MATCHING ALGORITHM FOR TEXT IMAGE COMPRESSION

1 Introduction

Since the majority of office documents are in printed form, symbol pattern-matching is an important technique in the automation of the office. The code generated includes an exact representation of each library symbol pattern, and compression is highly dependent on the final size of the library. Ideally, the library should contain exactly one pattern for each distinct symbol in the document. In practice, the library is usually much larger, containing many superfluous entries due to incorrect isolation and inefficient matching.

A good matching algorithm is therefore crucial not only to the accuracy but also to the compression efficiency of a symbol pattern-matching scheme. The matching criterion must be tight enough to prevent mismatches, while able to recognise matches between identical symbols distorted by digitization noise. In a general purpose text image coding system, the algorithm should perform well over a wide range of styles and sizes of text.

In the following, existing matching algorithms are described and their suitability for general-purpose text image compression is investigated. A new algorithm is formulated which exhibits an improved overall performance.

2 Existing matching algorithms

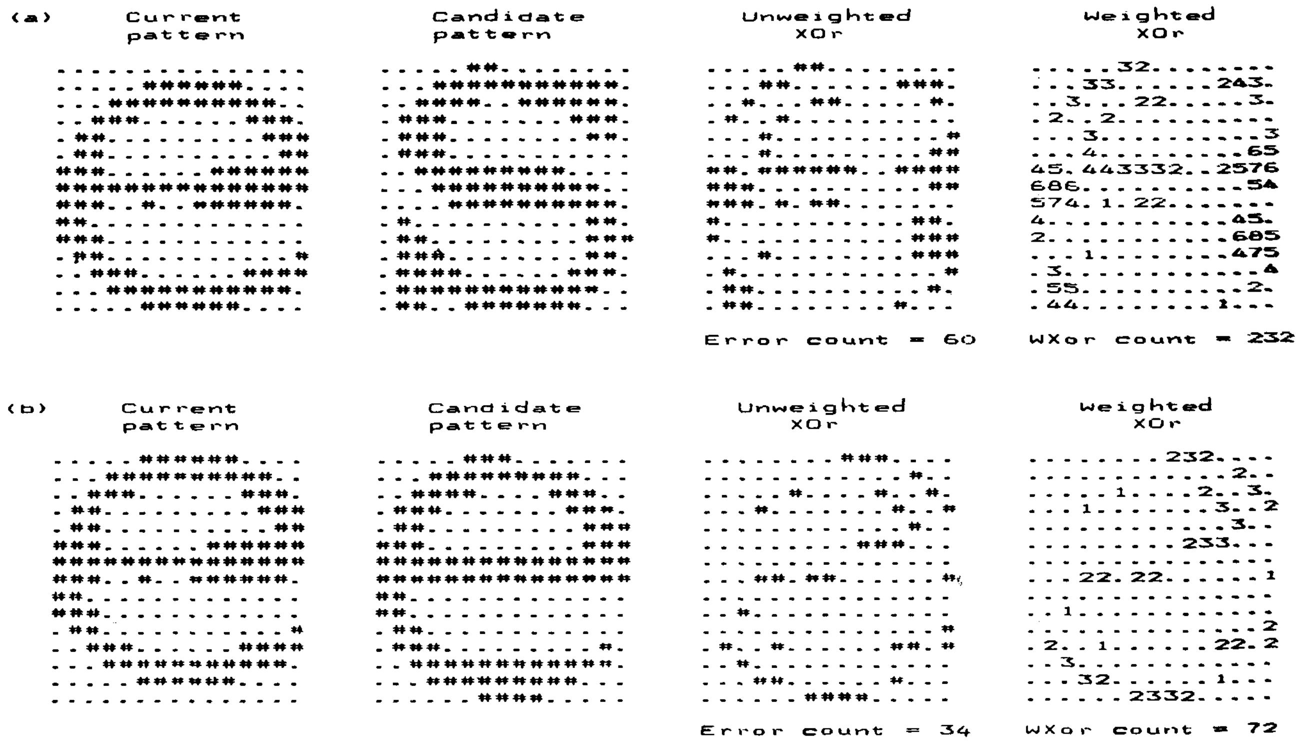
The matching algorithms in various reported schemes[23,25,26] make use of the Weighted Exclusive-Or (WXOr) error map of the two compared patterns. The WXOr map is obtained from the unweighted error (XOr) map by replacing each error with the error count in the surrounding 3×3 neighbourhood. In this way, greater weight is given to clustered errors which occur with distinct symbol patterns (Fig. 6a) than to sparse errors which can occur along the common boundary of similar symbol patterns (Fig. 6b).

In the Pattern Matching and Substitution (PMS) scheme[25], a matching decision is determined from the WXOr map using only local criteria. A match is rejected if either (1) any error pel has a weight of 5 or more, or (2) any error pel has both (i) two neighbouring error pels which are not adjacent (to each other) and (ii) a 3×3 neighbourhood in which all 9 pels are the same colour in one of the two symbol patterns. Rule 1 rejects most distinct patterns, but rule 2 is included for pairs of distinct symbols whose difference is manifested only in narrow strokes or gaps.

The Combined Symbol Matching (CSM) scheme[23] determines a match by comparing the sum of the weighted errors (the WXOr count) against a sliding scale of threshold values related to symbol size. The scale of threshold values is not quoted, but is said to be a non-linear function of the symbol's black pel count, obtained from an empirically determined look-up table. Another scheme[26] quotes threshold values for the CSM algorithm applied to text images sampled at half the vertical resolution (100 pels per inch). These values are not appropriate to images sampled at full resolution.

The threshold function must therefore be determined experimentally. This can be achieved by plotting the WXOr count against the joint black pel count, for pairs of similar and distinct symbol patterns in a training sample. The optimum threshold scale is an increasing function which eliminates matching errors (i.e. distinct symbol pairs resulting in a WXOr count within the threshold) and minimizes rejections (i.e. similar symbol pairs resulting in a WXOr count outside the threshold) in the training sample.

Both the PMS and CSM algorithms perform satisfactorily for samples of 10-pitch typewritten text. Other sizes and styles of text lead to problems, such as the example illustrated in Fig. 7. The PMS algorithm rejects a match between the pair of 'n's, but declares one between the 'a' and 's'. The CSM algorithm with a threshold low enough to reject the 'a' and 's' would also reject the two 'n's, whose WXOr count is higher.



```
(a)   Current           Candidate         Unweighted        weighted
      pattern           pattern              XOr               XOr

...............   .....##........   .....##........   .....32........
.....######....   ...###########.   ...##......###.   ...33......243.
...##########..   ..####..######.   ..#...##.....#.   ..3...22.....3.
..###......###.   .###.......###.   .#..#..........   .2..2..........
.##.........###   .###........##.   ...#..........#   ...3..........3
.##..........##   .###...........   ...#.........##   ...4.........65
###......######   ..#########....   ##.######..####   45.443332..2576
###############   ...##########..   ###..........##   686..........54
###..#..######.   ....##########.   ###.#.##.......   574.1.22.......
##.............   .#..........##.   #...........##.   4...........45.
###............   .##.........###   #...........###   2...........685
.##...........#   .###........##.   ...#........###   ...1........475
..###......####   .####......###.   .#............#   .3............4
...###########.   .############..   .##..........#.   .55..........2.
.....######....   .##..#######...   .##........#...   .44........1...

                                      Error count = 60  WXor count = 232

(b)   Current           Candidate         Unweighted        weighted
      pattern           pattern              XOr               XOr

.....######....   .....###.......   .........###...   ........232....
...##########..   ...#########...   ............#..   ............2..
..###......###.   ..####....###..   .....#....#..#.   .....1....2..3.
.##.........###   .###.......###.   ...#.......#..#   ...1.......3..2
.##..........##   .##.........###   ............#..   ............3..
###......######   ###.........###   .........###...   .........233...
###############   ###############   ...............   ...............
###..#..######.   ###############   ...##.##......#   ...22.22......1
##.............   ##.............   ...............   ...............
###............   ##.............   ..#............   ..1............
.##...........#   .##............   ..............#   ..............2
..###......####   .###.........#.   .#..#......##.#   .2..1......22.2
...###########.   ..############.   ..#............   ..3............
.....######....   ...#########...   ...##......#...   ...32......1...
...............   ......####.....   ......####.....   ......2332.....

                                      Error count = 34  WXor count = 72
```

Fig. 6 Error maps used in the PMS and CSM algorithms: (a) distinct symbol pair, (b) similar symbol pair

```
(a)  Current          Candidate        Unweighted       Weighted
     pattern          pattern             XOr              XOr

     ..###.....       ...##.....       ..#.......       ..1.......
     .######...       .#######..       .......#..       .......1..
     ##..#####.       .########.       #.##......       1.22......
     ##......#.       ##.....##.       .......#..       .......1..
     ##......#.       #.......#.       .#........       .4........
     ######....       ....#####.       ####.###.        3442.243.
     .#######..       .########.       ........#.       ........4.
     ....#####.       #####...#.       ########..       2333344..
     #......##.       #.......#.       .......#..       .......3..
     #......##.       #......##.       ..........       ..........
     ###.#####.       ##...#####       ..#.#...#        ..1.1...3
     .#######..       ###########      #......##        2......33
     ...##.....       .####.....       .##.......       .32.......

                                       Count = 31       Count = 81

(b)  Current          Candidate        Unweighted       Weighted
     pattern          pattern             XOr              XOr

     ....##...        ....##...        ..........       ..........
     .#######..       #########.       #......#.        2......2.
     .##...###        ###....##        #.....#..        3.....3..
     .#......#        ##.....##        #......#.        3......2.
     .#......#        ##......#        #........        3........
     .#......#        ##......#        #........        4........
     .#......#        #.......#        ##.......        55.......
     .#......#        #.......#        ##.......        66.......
     .#......#        #.......#        ##.......        66.......
     .#......#        #.......#        ##.......        66.......
     .#......#        #.......#        ##.......        66.......
     .#......#        #.......#        ##.......        44.......

                                    Count = 20       Count = 88
```

Fig. 7 Illustration of weakness of the PMS and CSM algorithms for small thin characters: (a) distinct symbol pair, (b) similar symbol pair

The poor performance of the existing algorithms in this example is partly due to the thinness of the characters. Both symbol patterns in Fig. 7b have vertical strokes just one pel in thickness, but the separation of the vertical strokes differs by one pel. It is not possible to superimpose these two patterns such that both vertical strokes coincide. The shift of one vertical stroke between the two patterns results in a vertical band two pels thick in the error map. This concentration of error pels results in artificially high weighted errors, causing both the PMS and CSM algorithms to fail in this instance.

Another problem with the CSM algorithm is illustrated in Fig. 8. The error count for the two 'm's is higher than that for the bolder 'a' and 's', even though the latter pair have a higher black-pel count. Thus an increasing threshold function which distinguishes the 'a' from the 's' would reject a match between the two 'm's. This arises because thicker characters inevitably have high black pel counts, but do not necessarily result in higher WXOr counts.

3 Proposed matching algorithm

The proposed algorithm[31,32], which is intended to overcome the disadvantages of the existing algorithms mentioned above, is similar to the CSM algorithm, but with two major modifications.

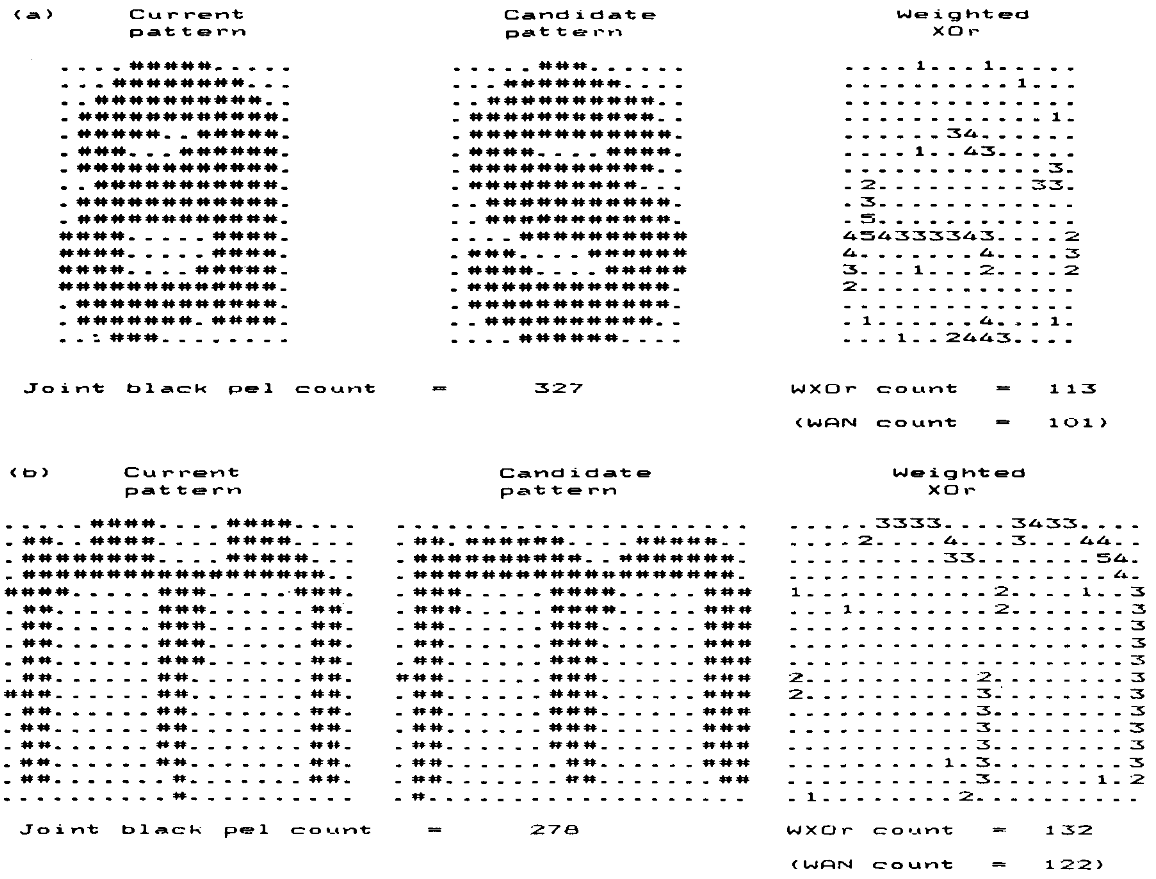
Firstly, the weighted errors are evaluated in a different way, which reduces the problem encountered with thin characters. The two symbol patterns are superimposed, and elements which differ are indicated in an error map. Each error element is then tagged in one of two ways, according to which of the two symbol patterns has a black pel in the corresponding position. Examples of these error maps are given in Fig. 9, where '+' denotes pels which are black in the first symbol but not the second, and '−' denotes error pels which are black in the second symbol but not the first.

In a computer, this tagged error map is more conveniently represented as two primary matrices: E1, in which '1's represent '+' errors, and E2, in which '1's represent '−' errors. If the two symbol patterns are held in binary matrices S1 and S2, having identical dimensions, then E1 and E2 can be obtained using bitwise logical operations, from

$$E1 = S1 \;\&\; \overline{S2}$$
$$E2 = \overline{S1} \;\&\; S2$$

Hence the tagged error map defined in this way is called an And-Not map.

The weighted And-Not (WAN) map is then obtained as follows. Each '−' error is given a weight equal to the number of '−' errors in a 3×3 neighbourhood centred on the element in question. Similarly, each '+' error is weighted according to the number of neighbouring '+' errors. Elements not tagged in the And-Not map are given zero weight. The WAN errors are



```
(a)     Current              Candidate            Weighted
        pattern              pattern                XOr

   ....#####.....       .....###......       ....1...1.....
   ...########...       ...#######....       ..........1...
   ..##########..       ..##########..       ..............
   .############.       .###########..       ............1.
   .#####..#####.       .############.       ......34......
   .###...######.       .####....####.       ....1..43.....
   .############.       .###########..       ............3.
   ..###########.       .##########...       .2.........33.
   .############.       ..###########.       .3............
   .############.       ..###########.       .5............
   ####.....####.       ....##########       454333343....2
   ####.....####.       .###....######       4.......4....3
   ####....#####.       .####....#####       3...1...2....2
   #############.       .############.       2.............
   .############.       .############.       ..............
   .#######.####.       ..##########..       .1......4...1.
   ..:###........       ....######....       ...1..2443....

 Joint black pel count  =   327             WXOr count  =  113

                                            (WAN count  =  101)
```

```
(b)      Current                  Candidate                  Weighted
         pattern                  pattern                      XOr

.....####....####....   .....................   .....3333....3433....
.##..####....####....   .##.######....#####..   ....2....4...3...44..
.########....#####...   .##########..#######.   .........33.......54.
.##################..   .###################.   ...................4.
####.....###.....###.   .###.....####.....###   1...........2....1..3
.##......###......##.   .###.....####.....###   ...1........2.......3
.##......###......##.   .##......###......###   ....................3
.##......###......##.   .##......###......###   ....................3
.##......###......##.   .##......###......###   ....................3
.##......##.......##.   ###......###......###   2..........2........3
###......##.......##.   .##......###......###   2..........3........3
.##......##.......##.   .##......###......###   ...........3........3
.##......##.......##.   .##......###......###   ...........3........3
.##......##.......##.   .##......###......###   ...........3........3
.##......##.......##.   .##.......##......###   .........1.3........3
.##.......#.......##.   .##.......##.......##   ...........3......1.2
..........#..........   .#...................   .1........2..........

 Joint black pel count  =   278                 WXOr count  =  132

                                                (WAN count  =  122)
```

Fig. 8 Illustration of weakness of the CSM algorithm for small thin characters: (a) distinct symbol pair, (b) similar symbol pair

```
(a) Current       Candidate      Unweighted     Weighted
    pattern       pattern          AndNot        AndNot

    ..###....     ...##....      ..+......      ..1......
    .#####...     .######..      ......-..      ......1..
    ##..####.     .#######.      +.--.....      1.22.....
    ##.....#.     ##....##.      ......-..      ......1..
    ##.....#.     #......#.      .+.......      .4.......
    #####....     ....####.      ++++.---.      3442.243.
    .######..     .#######.      .......-.      .......3.
    ....####.     ####...#.      ----+++..      2332243..
    #.....##.     #......#.      ......+..      ......3..
    #.....##.     #.....##.      .........      .........
    ###.####.     ##...####      ..+.+...-      ..1.1...3
    .######..     #########      -......--      2......33
    ...##....     .####....      .--......      .32......

                                                Count = 77
```

```
(b) Current       Candidate      Unweighted     Weighted
    pattern       pattern          AndNot        AndNot

    ....##...     ....##...      .........      .........
    .######..     ########.      -......-.      2......1.
    .##...###     ###....##      -.....+..      3.....1..
    .#......#     ##.....##      -......-.      3......1.
    .#......#     ##......#      -........      3........
    .#......#     ##......#      -........      3........
    .#......#     #.......#      -+.......      32.......
    .#......#     #.......#      -+.......      33.......
    .#......#     #.......#      -+.......      33.......
    .#......#     #.......#      -+.......      33.......
    .#......#     #.......#      -+.......      33.......
    .#......#     #.......#      -+.......      22.......

                                             Count = 50
```

Fig. 9 Error maps used in the WAN algorithm: (a) distinct symbol pair, (b) similar symbol pair

then summed over the whole map, resulting in the WAN count. As in the CSM algorithm, the matching decision is then made by comparing the weighted error count against an empirically determined sliding scale of threshold values.

The essential effect of this modification is that, in the WAN map, two neighbouring error elements only increase each other's weights if both are due to black pels in the same symbol pattern. Consequently, for a cluster of errors which arises spuriously, perhaps due to a small shift in a thin stroke as in Figs. 7b and 9b, the weighted errors are often halved. On the other hand, in distinct symbol patterns where the errors in a cluster tend to be due to black pels in the same symbol (e.g. Figs. 7a and 9a), the WAN count is not significantly lower than the WXOr count. The WAN count is clearly a more consistent measurement of mismatch in these examples.

The second modification tackles the problem associated with thicker characters. In the CSM algorithm, the threshold scale is an increasing function of the symbol's black pel count. Because thicker characters have high black pel counts in proportion to their size, a threshold scale which correctly rejects a match between thick characters such as in Fig. 8a often results in the rejection of a pair of similar symbols with a lower black pel count, such as in Fig. 8b.

In the modified algorithm, the threshold scale is a function of an estimate of the symbol perimeter length, rather than of the black pel count. The reasoning behind this is that when two similar symbol patterns are superimposed, the matching errors are randomly distributed along the inner and outer perimeters of the symbols. It therefore follows that a threshold scale which is an increasing function of the perimeter length will recognise a greater proportion of similar symbol pairs.

Although the direct measurement of a symbol's perimeter length from its binary pattern is computationally intractable, a simple estimate is obtainable from four easily measurable parameters of the symbol pattern. These are the overall width and height of the symbol pattern, and the numbers of internal white runs (i.e. bounded at both ends by black pels) in (a) a horizontal scan, and (b) a vertical scan, of the symbol pattern. Fig. 10 shows that twice the sum of these four parameters is equal to the number of exposed faces of black edge pels, which is a good approximation of the perimeter length.

For the symbol pairs in Figs. 8a and 8b the estimated perimeter lengths obtained by this method are 186 and 260 respectively, i.e. higher for the similar symbols than for the distinct symbols. Thus a threshold scale, which is an increasing function of the symbol perimeter estimate, can be chosen so that the WAN algorithm gives correct matching decisions for both symbol pairs in this example.

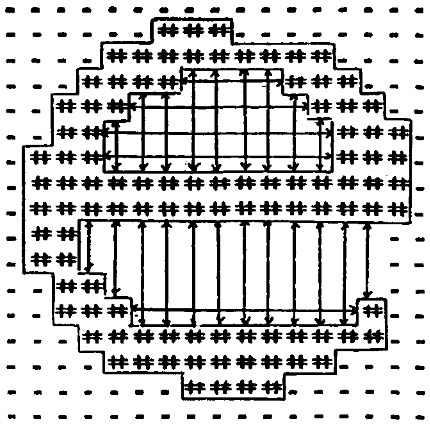


Fig. 10 Evaluation of symbol perimeter estimate

Symbol perimeter estimate
= number of exposed sides of black edge pels
= 2 × width + 2 × height
+ 2 × number of horizontal internal white runs
+ 2 × number of vertical internal white runs
= 2 × 15 + 2 × 15 + 2 × 5 + 2 × 21
= 112

4 Simulation

To test and evaluate the modified algorithm, a symbol matching scheme is simulated in Pascal on an ICL Perq 1 computer running under POS. The various activities in the simulation are flowcharted in Fig. 11 and are described in detail below. The simulation follows the CSM scheme[23] in all but two stages of the process. One exception is the crucial matching stage, where the PMS algorithm[25] and the proposed WAN algorithm are implemented as optional alternatives to CSM. The other exception is the symbol coding stage, where the codes used by Johnsen et al[25] have proved more efficient.

(a) Symbol extraction

For the purposes of extraction from the image, a symbol is defined to be any connected black object which can be entirely contained within a square window whose sides measure 32 pels (or about 4 mm). Each symbol is isolated by boundary-following and copied to another part of memory. The symbol is then deleted from the memorized image by performing a logical Exclusive-OR with the copy of the symbol. The copy is then passed to the screening and matching processes.

(b) Candidate screening

In order to minimize the number of applications of the time-consuming matching process, a number of basic features of the current symbol are

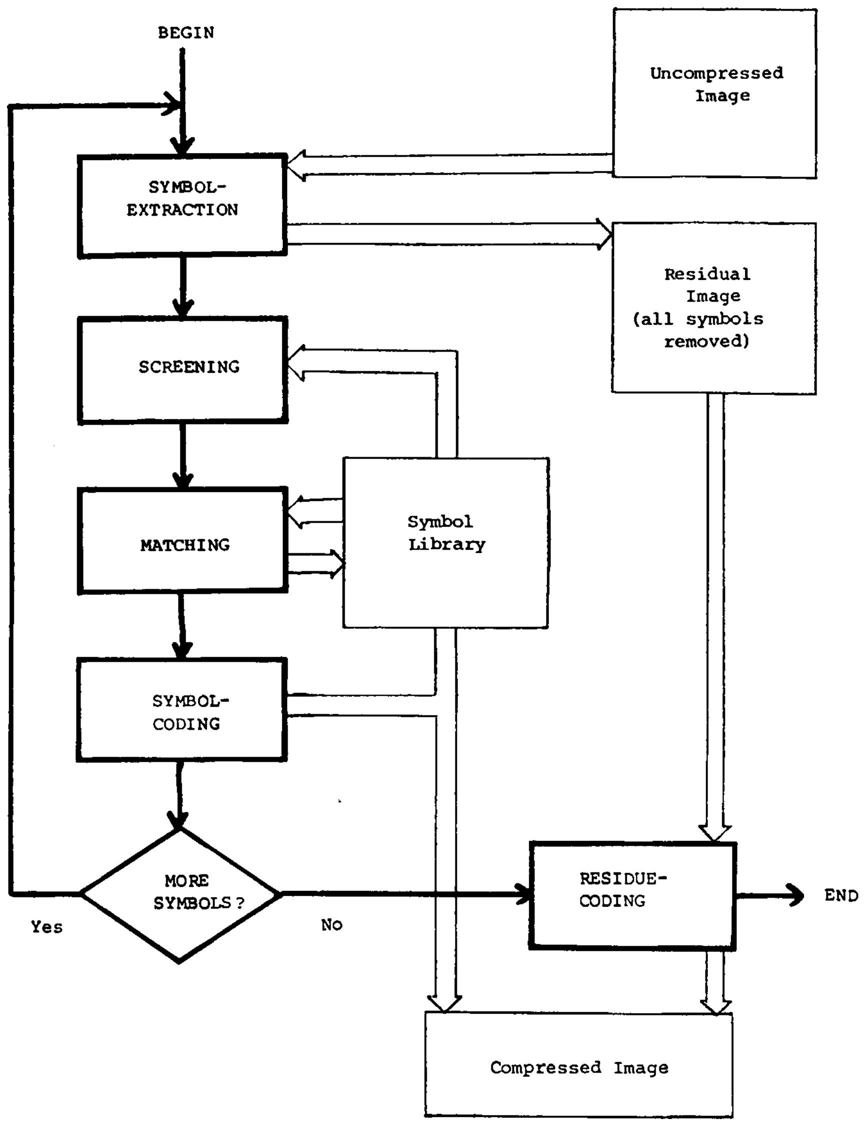


Fig. 11 Flowchart for the simulated symbol pattern-matching and coding scheme

measured and compared with those of the library symbols. Only library symbols whose features are sufficiently close to those of the current symbol are selected for matching. These are then sorted so that the symbols most likely to result in a successful match are processed first.

The features used for screening are, conveniently, the same four parameters used to estimate the symbol perimeter length as described above. Two symbols are considered likely candidates for matching if their widths and heights differ by no more than 2 pels, and their numbers of internal white runs in either direction differ by no more than 5.

(c) Symbol matching

Those library symbols which pass the screening process are matched against the candidate symbol by the following procedure: first the two patterns are registered against one another so that the distance between the corresponding block edges does not exceed 1 pel. There may be as few as one or as many as nine ways of doing this, depending on how similar are the block dimensions of the two symbols. The most central registration of the two patterns is taken first.

For each registration, the selected algorithm (PMS, CSM or WAN) is applied to determine whether the two patterns match. If a match is found and there are other registrations of the patterns still to be tried, the unweighted error count is noted for the matching position. The full matching algorithm will not be applied again to that pair of symbols unless a lower unweighted error count is found. If no match is identified, the algorithm is repeated for all possible registrations of the two symbols.

(d) Library maintenance

When no matching symbol is identified, the current symbol is added to the library. The library contains the full bit-map representation of each symbol pattern, up to a maximum of 192 K bits, together with a table giving the bit address of each pattern, its screening features, and the number of times it has been used, up to a maximum of 512 symbols. If either of these limits is reached, one or more of the least used library symbols is deleted to make room for each new symbol.

(e) Symbol coding

Variable length codewords are used to define the starting position and library identification of each symbol, with special codes for 'new symbol', 'same symbol', and 'no more symbols (on current scan line)'. The patterns of new library symbols are encoded by a variation of the MRC algorithm.

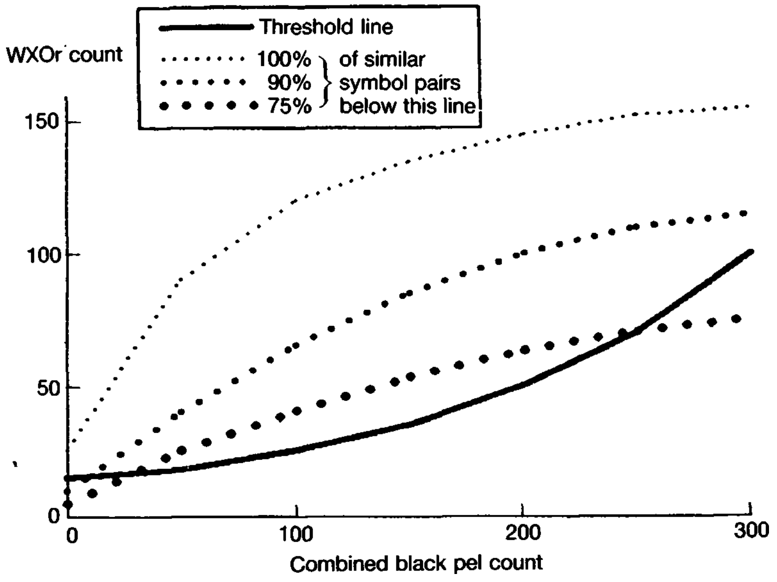
(f) Residue coding

The residual image, from which all symbols have been deleted, is encoded using the MRC-II algorithm[8].

5 Results

5.1 Training

In order to establish a threshold function which could be globally applied to a wide range of sizes and styles of text, both the CSM and WAN algorithms were trained on a special set of input data. The training set consists of samples of six different character sets, with symbol widths between 1 and 4 mm, and in several different fonts.



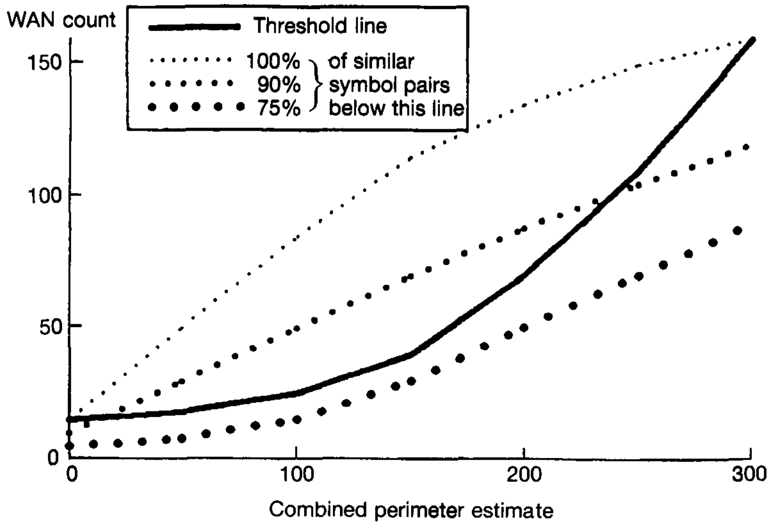


Fig. 12 Threshold function plots obtained from training sample, for: (a) the CSM algorithm, (b) the WAN algorithm

Using the screening algorithm described above, pairs of symbol patterns were selected from the training set, and the minimum WXOr and WAN counts for each pair were calculated. The combined black-pel and symbol-perimeter counts for each pair were also obtained. A threshold function for the CSM algorithm was obtained from a plot of the WXOr count against the combined black-pel count (Fig. 12a). A threshold function for the WAN algorithm was obtained from a plot of the WAN count against the combined symbol perimeter estimate (Fig. 12b). In both cases the threshold function was the highest curve which could be drawn below all of the distinct symbol pair counts.

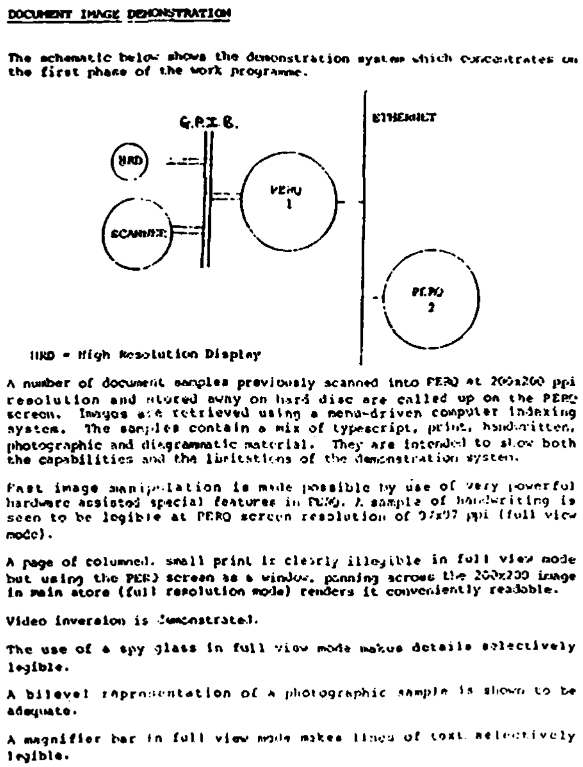
Figs 12a and 12b clearly show that the WAN threshold cuts off fewer of the similar symbol pairs than the CSM threshold. The overall rejection rates for the training sample (i.e. the proportions of similar symbol pairs not recognised by each matching algorithm) are 36% for CSM and 24% for WAN.

5.2 Compression

To evaluate the performance of the new algorithm, the simulation was applied to two A4-size documents shown in Fig. 13. The DRAFT document consists of typical typewritten symbols averaging 2 mm in width, together with some graphics. The SUBSID document consists of 'san serif' text, in three different sizes and also of varying boldness. Coding of these documents was simulated, using first the MRC-II exact coding algorithm, and then by symbol matching, using in turn the PMS, CSM and WAN algorithms.

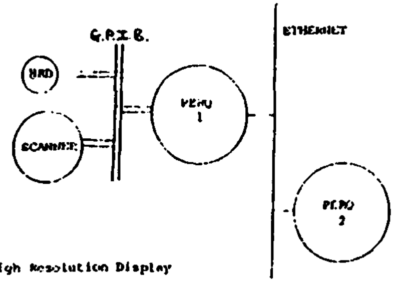
Table 1 Compression results

Document	Matching algorithm	Symbols in library	Compression factor	Matching errors
DRAFT typewritten 1161 symbols some graphics	MRC-II	–	18.7	–
	PMS	101	55.7	0
	CSM	202	43.5	0
	WAN	124	54.4	0
SUBSID san serif various text sizes	MRC-II	–	13.0	0
	PMS	199	45.1	16 (0.51%)
2865 symbols	CSM	351	36.4	2 (0.07%)
	WAN	270	41.3	1 (0.03%)



DOCUMENT IMAGE DEMONSTRATION

The schematic below shows the demonstration system which concentrates on the first phase of the work programme.



HRD = High Resolution Display

A number of document samples previously scanned into PERQ at 200x200 ppi resolution and stored away on hard disc are called up on the PERQ screen. Images are retrieved using a menu-driven computer indexing system. The samples contain a mix of typescript, print, handwritten, photographic and diagrammatic material. They are intended to show both the capabilities and the limitations of the demonstration system.

Fast image manipulation is made possible by use of very powerful hardware assisted special features in PERQ. A sample of handwriting is seen to be legible at PERQ screen resolution of 97x97 ppi (full view mode).

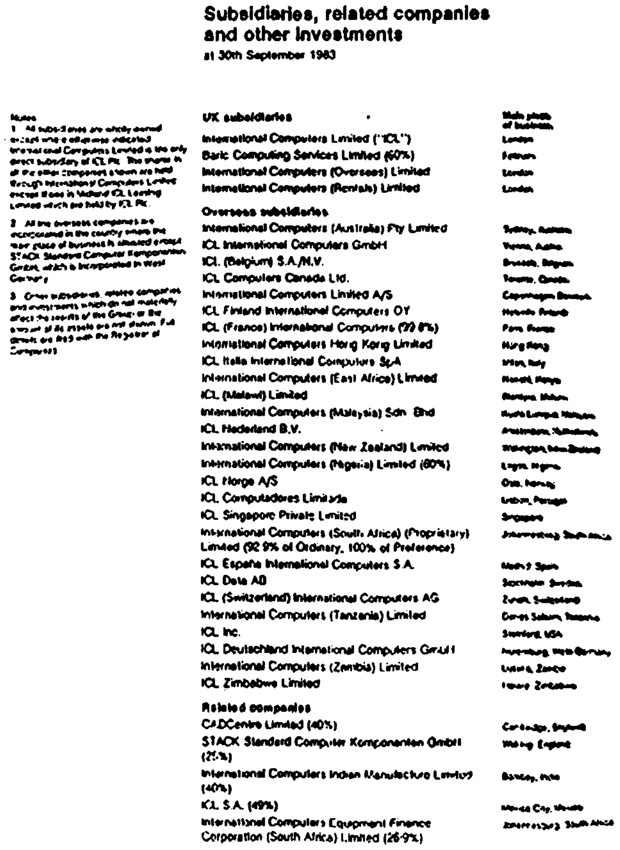
A page of columned, small print is clearly illegible in full view mode but using the PERQ screen as a window, panning across the 200x200 image in main store (full resolution mode) renders it conveniently readable.

Video inversion is demonstrated.

The use of a spy glass in full view mode makes details selectively legible.

A bilevel representation of a photographic sample is shown to be adequate.

A magnifier bar in full view mode makes lines of text selectively legible.



Subsidiaries, related companies and other investments
at 30th September 1983

Notes
1 All subsidiaries are wholly owned except where otherwise indicated. International Computers Limited is the only direct subsidiary of ICL Plc. The shares in all the other companies shown are held through International Computers Limited except those in Midland ICL Leasing Limited which are held by ICL Plc.

2 All the overseas companies are incorporated in the country where the main place of business is situated except STACK Standard Computer Komponenten GmbH, which is incorporated in West Germany.

3 Certain subsidiaries, related companies and investments which do not materially affect the results of the Group or the amount of its assets are not shown. Full details are filed with the Register of Companies.

UK subsidiaries	Main place of business
International Computers Limited ("ICL")	London
Baric Computing Services Limited (60%)	Feltham
International Computers (Overseas) Limited	London
International Computers (Rentals) Limited	London
Overseas subsidiaries	
International Computers (Australia) Pty Limited	Sydney, Australia
ICL International Computers GmbH	Vienna, Austria
ICL (Belgium) S.A./N.V.	Brussels, Belgium
ICL Computers Canada Ltd.	Toronto, Canada
International Computers Limited A/S	Copenhagen, Denmark
ICL Finland International Computers OY	Helsinki, Finland
ICL (France) International Computers (99.8%)	Paris, France
International Computers Hong Kong Limited	Hong Kong
ICL Italia International Computers SpA	Milan, Italy
International Computers (East Africa) Limited	Nairobi, Kenya
ICL (Malawi) Limited	Blantyre, Malawi
International Computers (Malaysia) Sdn. Bhd	Kuala Lumpur, Malaysia
ICL Nederland B.V.	Amsterdam, Netherlands
International Computers (New Zealand) Limited	Wellington, New Zealand
International Computers (Nigeria) Limited (60%)	Lagos, Nigeria
ICL Norge A/S	Oslo, Norway
ICL Computadores Limitada	Lisbon, Portugal
ICL Singapore Private Limited	Singapore
International Computers (South Africa) (Proprietary) Limited (92.9% of Ordinary, 100% of Preference)	Johannesburg, South Africa
ICL España International Computers S.A.	Madrid, Spain
ICL Data AB	Stockholm, Sweden
ICL (Switzerland) International Computers AG	Zurich, Switzerland
International Computers (Tanzania) Limited	Dar es Salaam, Tanzania
ICL Inc.	Stamford, USA
ICL Deutschland International Computers GmbH	Augsburg, West Germany
International Computers (Zambia) Limited	Lusaka, Zambia
ICL Zimbabwe Limited	Harare, Zimbabwe
Related companies	
CADCentre Limited (40%)	Cambridge, England
STACK Standard Computer Komponenten GmbH (25%)	Woking, England
International Computers Indian Manufacture Limited (40%)	Bombay, India
ICL S.A. (49%)	Mexico City, Mexico
International Computers Equipment Finance Corporation (South Africa) Limited (26.9%)	Johannesburg, South Africa

Fig. 13 Test documents: (a) DRAFT, (b) SUBSID

The results of this are summarised in Table 1. The highest compression is generated by the PMS algorithm, but this causes an unacceptable number of matching errors in the SUBSID document. The CSM algorithm, using the globally trained threshold function, results in a much lower error rate, but at the cost of a large increase in library size and hence a considerable reduction in compression. Only the proposed WAN algorithm succeeds in almost eliminating matching errors without seriously reducing the compression in either example. In both examples, compression roughly three times that of MRC-II is achieved using the new algorithm.

6 Discussion

Although further testing is needed to prove the effectiveness of the new algorithm, the preliminary results indicate a clear improvement over existing matching algorithms applied to the compression of arbitrary printed documents. The extremely low error rates observed would be acceptable in many applications. If greater accuracy were required, the error rate could be reduced further by training the algorithm on a larger sample.

Acknowledgement

The authors wish to thank ICL Systems Strategy Centre, Bracknell, England, for sponsorship of the original work described in this paper.

References

1 ROTHGORDT, U.: 'Run-length coding method for black and white facsimile with a ternary code as intermediate step', *Electronics Letters*, 1975, Vol. 11, pp. 101–102.
2 DE COULON, F. and JOHNSEN, O.: 'Adaptive block scheme for source coding of black and white facsimile', *Electronics Letters*, 1976, Vol. 12, pp. 61–62.
3 JAPAN: 'Proposal for draft recommendation of two-dimensional coding scheme', CCITT SG XIV Document 42, 1978.
4 MUSMANN, H.G. and PREUSS, D.: 'Comparison of redundancy reducing codes for facsimile transmission of documents', *IEEE Trans. Commun.*, 1977, Vol. COM-25, pp. 1425–1432.
5 HUNTER, R. and ROBINSON, H.: 'International digital facsimile coding standards', *Proc. IEEE*, 1980, Vol. 68, pp. 854–867.
6 BODSON, D. and SCHAPHORST, R.: 'Compression and error-sensitivity of two-dimensional facsimile coding techniques', *Proc. IEEE*, 1980, Vol. 68, pp. 846–853.
7 LANGDON, G.G. and RISSANEN, J.: 'Compression of black and white images with arithmetic coding', *IEEE Trans. Commun.*, 1981, Vol. COM-29, pp. 858–867.
8 BODSON, D., DEUTERMANN, A.R., URBAN, S.J. and CLARKE, C.E.: 'Measurement of data compression in advanced group-4 facsimile systems', *Proc. IEEE*, 1985, Vol. 73, pp. 731–739.
9 TAKAGI, M. and TSUDA, D.: 'Bandwidth compression for facsimile using signal modification', *Electronics and Comms. in Japan*, 1977. Vol. 50A, Pt. 2, pp. 9–16.
10 TING, D. and PRASADA, B.: 'Digital preprocessing techniques for encoding of graphics', *Proc. IEEE*, 1980, Vol. 68, pp. 757–769.
11 BILLINGS, T.L.: 'Pre-transmission enhancement of facsimile images', *IBM Tech. Disclosure Bull.*, 1979, Vol. 21, pp. 3086–3087.
12 ISMAIL, M.G.B. and CLARKE, R.J.: 'New pre-processing technique for digital facsimile transmission', *Electronics Letters*, 1980, Vol. 16, pp. 355–356.

13 JUDD, I.D.: 'Compression of binary images by stroke encoding', *IEE J Comput. Digital Techniques*, 1979, Vol. 2, pp. 41–48.
14 USUBUCHI, T., MIZUNO, S. and IINUMA, K.: 'Efficient facsimile signal data reduction by using a thinning process', Proc. NTC, 1977, pp. 49.2.1–49.2.6.
15 MORRIN, T.H.: 'Chain-link compression of arbitrary black-white images', Computer Graph. Image Process., 1976, Vol. 5, pp. 172–184.
16 ARPS, R.B.: 'Binary Image Compression', in 'Image Transmission Techniques', Vol. 12 (W.K. Pratt, Ed.), Academic Press, New York, 1979, Chap. 7.
17 HOLT, M.J.J. and XYDEAS, C.S.: 'Compression technique for handwritten and graphic images', in preparation.
18 PAVLIDIS, T. and CHERRY, L.L.: 'Vector and arc encoding of graphics and text', Proc. IEEE VI Int. Joint Conf. Pattern Recognition, 1982, pp. 489–491.
19 PAVLIDIS, T.: 'Curve fitting as a pattern recognition problem', Proc. IEEE VI Int. Joint Conf. Pattern Recognition, 1982, pp. 853–859.
20 RAMACHANDRAN, K.: 'Coding method for vector representation of engineering drawings', *Proc. IEEE*, 1980, Vol. 68, pp. 813–817.
21 ASCHER, R.N. and NAGY, G.: 'A means for achieving a high degree of compaction on sign-digitized text', *IEEE Trans. Comput.*, 1974, Vol. C-23, pp. 1174–1179.
22 MANTAS, J.: 'A survey of character recognition methodologies', *Proc. MELECON*, 1985, Vol. II, pp. 267–271.
23 PRATT, W.K., CAPITANT, P.J., CHEN, W., HAMILTON, E.R. and WALLIS, R.H.: 'Combined Symbol matching facsimile data compression system', *Proc. IEEE*, 1980, Vol. 68, pp. 786–796.
24 BRICKMAN, N.F. and ROSENBAUM, W.S.: 'Word auto-correlation redundancy match (WARM) technology', IBM J. Res. Develop., 1982, Vol. 26, pp. 681–686.
25 JOHNSEN, O., SEGEN, J. and CASH, G.L.: 'Coding of two-level pictures by pattern-matching and substitution', *Bell System Tech. J.*, 1983, Vol. 62, pp. 2513–2547.
26 SILVER, D.M. and JOHNSON, D.A.H.: 'Facsimile coding using symbol matching techniques', *IEE Proc.*, 1984, Vol. 131-F, pp. 125–129.
27 USUBUCHI, T., OMACHI, T. and IINUMA, K.: 'Adaptive predictive coding for newspaper facsimile', *Proc. IEEE*, 1980, Vol. 68, pp. 807–813.
28 WONG, K.Y., CASEY, R.G. and WAHL, F.M.: 'Document analysis system', *IBM J. Res. Develop.*, 1982, Vol. 26, pp. 647–656.
29 OKAMOTO, N., NAKAMURA, O. and MINAMI, T.: 'Character segmentation for mixed-mode communication', Proc. IFIP 9th World Computer Congress, Elsevier, 1983, pp. 681–685.
30 POSTL, W.: 'Halftone recognition by an experimental text and facsimile workstation', Proc. VI Int. Joint Conf. Pattern Recognition, 1982, pp. 489–491.
31 HOLT, M.J.J. and XYDEAS, C.S.: 'Compression of bi-level text image data by symbol matching', Proc. Int. Conf. Advances in Image Processing and Pattern Recognition, Pisa, Dec. 1985, North Holland, pending publication.
32 HOLT, M.J.J.: 'Symbol pattern matching', British Patent Application No. 8525509, filed Oct. 1985.

Message structure as a determinant of message processing system structure

D.J. Ackerman
STC Network Systems Division, Kidsgrove, Staffordshire

Abstract

The paper is concerned with nested, or layered, multiplexing systems of the kind represented by layers 1–4 of the ISO Reference Model for Open Systems Interconnection. It shows that for these systems a direct relationship exists between the structure of a system and that of the messages requiring to be operated on by the system. It is shown that this relationship holds not only for the system as a whole but also for the individual layers and sub-layers of a system considered in relation to the structure of the layer and sub-layer protocol information fields that go to make up the messages.

It also shows that the generally accepted notion that structure is an expensive luxury is valid only because the importance of the relationship between system structure and message structure is not generally recognised. Given that these structures are defined to be mutually supporting, then the system attributes that come with structure can be had with a cost bonus.

1 Introduction

Previous articles[1,2] have appeared in this journal dealing with the ICL Information Processing Architecture, IPA, and its relationship to the ISO reference model for Open Systems Interconnection (OSI)[3]. This article is concerned with layers 1–4 of the OSI model, the functions of which collectively go to make up the IPA Telecommunications Function. More specifically, it is concerned with the layering principle as it relates to these layers and the effects on real implementations of its adoption as a modelling principle.

The ISO reference model standard gives the following assurances with respect to the relationship of the model to real-world systems:

- its scope covers only the externally visible behaviour of open systems;
- this behaviour is determined by the layer protocols to which an open system is required to conform;

- it is concerned to define the internal structure of an entirely abstract open system;
- the model is implementation independent in that it does not constrain the internal structure of a real open system.

However, in spite of these assurances, it is quite clearly the case that only pathologically deformed Open Systems will fail to reflect, in a real sense, the structure of the model. Only by reflecting its structure can the layer entities of a real open system hope to exhibit the property of layer-independence fundamental to the model. It is also clear that, no matter how a conforming real open system may be implemented, the messages requiring to be generated and processed by the system will have structures that reflect that of the model, not that of the system if different from the model. It may therefore be taken that underlying the assurances given by the ISO standard is the basic assumption that no necessary relationship exists between the internal structure`of a real open system and the structure of the information the system is called upon to process.

This paper challenges the validity of this assumption. It takes the example of a simple nested, or multilayered, multiplexing system implementing an extended form of Balanced High-Level Data Link Control (HDLC) procedures[4] at all layers. It explores in increasing levels of detail the relationships between the structure of the system and the structure of the messages operated on by the system and considers the effects on implementation of imposing message structures that do not reflect a hypothesised system structure. It is shown that implementation structure and message structure must be seen as two faces of the same coin; that if they are not so seen, then, in practice, the second will impress itself on the first.

At the final level of detail the idealised HDLC message header structures (formats) needed to support an idealised HDLC protocol machine structure are identified. These are compared with the formats that were chosen in practice for ISO standardisation. It is again shown how the machine structure implicit in the standardised formats imposes itself on the structure of an implementation; this precluding all possibility of realising in practice an ideally structured and least-cost implementation.

It is believed that the conclusions of the paper as they arise out of a case study of HDLC procedures are equally applicable to most other OSI Layer 1–4 protocols that have been, or are in the course of being, developed.

2 Multiplexing System

Fig. 1 illustrates the system to be considered.

In this illustration P, Q, R, etc. represent physically separate multiplexing/demultiplexing elements. Each element at one end of the system has its complementary peer element at the other. All peer elements communicate with one another using extended and idealised Balanced Mode HDLC procedures. The protocol extension is that needed to facilitate multiplexing of higher layer links through the use of the address (A) field of the standard HDLC Frame Format[5].

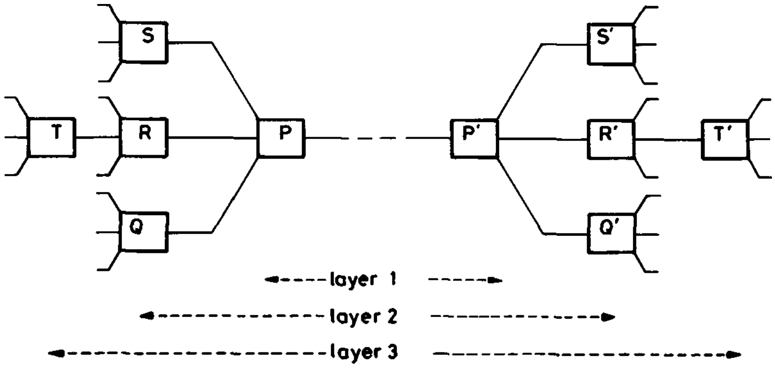


Fig. 1 The example multiplexing system

This system will now be considered, first at the level of structural detail illustrated in Fig. 1, and then at progressive levels of magnification of the structure of the individual multiplexing/demultiplexing elements.

3 Stage 1: the unmagnified system

Systems of the type of Fig. 1 can be constructed without difficulty to exhibit the properties of layer and lateral independence. These properties are defined for the purposes of this paper as follows:

- *layer independence:* the property enabling peer entities at any layer to be designed without knowledge of the details of the protocols used to effect communication between peer entities at higher and lower layers.

  This is the fundamental property ascribed to the layers of the ISO reference model.

- *lateral independence:* the property enabling pairs of corresponding peer entities at any one layer to be designed without knowledge of the details of the protocols used to effect communication between other pairs of corresponding peer entities at the same layer.

  This property is not dealt with in any great degree of detail by the reference model but is one of equal significance.

When the system of Fig. 1 is constructed to exhibit both properties, the structure of the messages transmitted between ends of the system, as they might be monitored on the communications media, will be as illustrated in Fig. 2.

This is the message structure as it will naturally, and necessarily, be generated by one end for processing at the other. It has the property of exactly reflecting the structures, taken as a whole, of the sending and receiving ends of the multiplexing system.

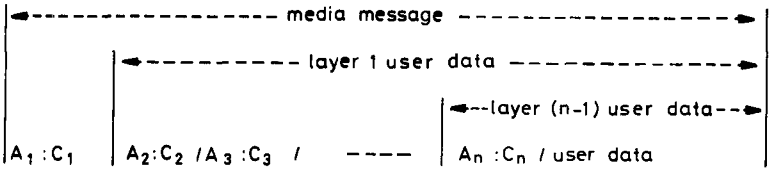


Fig. 2 Media message format corresponding to Fig. 1 system

Key: A_x:C_x = address and control fields of the HDLC frame format operated at layer x
n = a variable dependent on the level of entry of user data into the system

Suppose now that the following variant on the message structure were to be imposed *a priori* as a system design constraint:

$$A_1/A_2/A_3/\ldots/A_n : C_1/C_2/C_3/\ldots/C_n/\text{data}$$

On the face of it this might be seen as something of a rationalisation of the Fig. 2 format and as such a highly desirable constraint to impose. If imposed, then it could be handled in either of the following ways:

1 by introducing message format (syntax) transformation functions at the two ends of the line as the lowest level functional element in the structure; or
2 by embodying a knowledge of the message format in all layers.

The consequences of 1 are:

– a necessary increase in message processing cost in proportion to the complexity of the transformation function; the transformation given above as an example being a relatively trivial one compared with those actually encountered in practice;
– a necessary loss of the attributes of layer and lateral independence.

The loss in the second case follows from the fact that a knowledge of both the structure of the system and that of the message must be designed into the transformation function. No element in the structure can be modified without reference to this function. It becomes the key element binding the structure. All other elements become subservient to it. It becomes: the seat of systems intelligence; the fount of all knowledge; the governing element of the structure without which no other element can function.

The consequences of 2 are even more undesirable. Because systems intelligence is distributed it becomes, in each of its parts, unchangeable without reference to the sum of its parts. The resulting structure becomes no more modular, in the sense that entities can be modified without impact on other entities, than a monolith. The system becomes structured only in the sense that it is made up of bits and pieces. Fig. 1, intended as a representation of both the physical and functional structures of the system, becomes truly representative only of the first. It misleadingly represents the second, the most important elements of which are those exercising systems intelligence and management responsibility; the complexity of the second being magnified by the fact of distribution of the first.

It is reasonable to conclude that a failure to realise the attributes of layer and lateral independence will, in practice, tend to magnify rather than reduce costs. Further, that, at the level of magnification being considered, the achievement of these attributes is dependent on the degree to which real system structure is reflected in the message structures requiring to be operated on by the system.

4 Stage 2: internal structure of a single layer

The internal structure of a single multiplexing layer of the multiplexing system being considered may be designed to conform to one or other of the following structures;

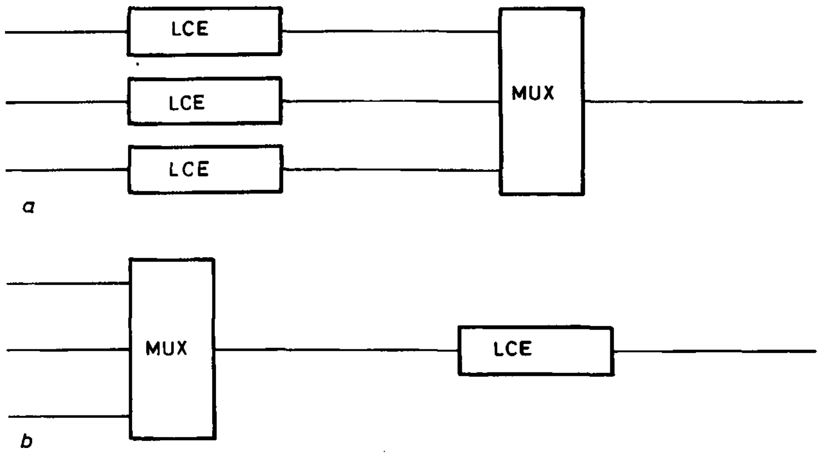


Fig. 3 Alternative sublayer structures: (*a*) individual link control; (*b*) common link control

Key: MUX =	multiplexing entity operating on an A (address) component of the line message
LCE =	link control entity operating on the related C (control) component of the line message

Now, if sublayer independence is to be achieved, then the order of the related A and C components of the message of Fig. 2 should be determined by the choice made between these structures. For individual link control the order will be

A : C/data

For common link control it will be

C : A/data

Consider the individual link control case. If a pathological implementation transposing the sublayers is attempted at one end, then a necessary feature of the implementation will be the introduction of fan-in/fan-out mechanisms, Y and X, above and below the link control sublayer as illustrated in Fig. 4.

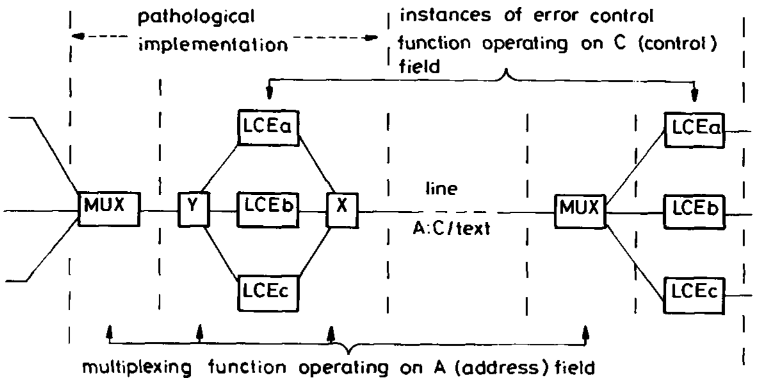

pathological implementation
instances of error control function operating on C (control) field
LCEa
LCEb
LCEc
MUX
Y
X
line
A:C/text
MUX
LCEa
LCEb
LCEc
multiplexing function operating on A (address) field

Fig. 4 System structure resulting from pathological implementation

Quite clearly, the mechanisms X and Y are wholly concerned with operating on the A field of the format. The sublayers, therefore, must of necessity lose their independence.

Furthermore, the following equalities must be seen within the pathological implementation:

function X = function MUX = complement of function Y

Given these equalities, then it may be further seen that function MUX and function Y, in effect, cancel one another, leaving only function X as the function visible to the nonpathological remote end MUX peer entity.

The general conclusion may be drawn that, at the level of detail being considered, the protocol machine structure required to process the message header information relating to a layer is implicit in the structure of the header information. This implied structure will impose itself on the machine structure in some form or another no matter what attempts may be made to escape from it. Further, given that header structure transformation functions are introduced to reconcile different machine and header structures, then the same cost and other penalties will be incurred for the layer as were incurred at stage 1 for the system as a whole.

5 Stage 3: internal structure of a link control entity

Considering now a link control sublayer entity of either Fig. 3*a* or 3*b*. This may either be designed as a pair of laterally independent simplex links

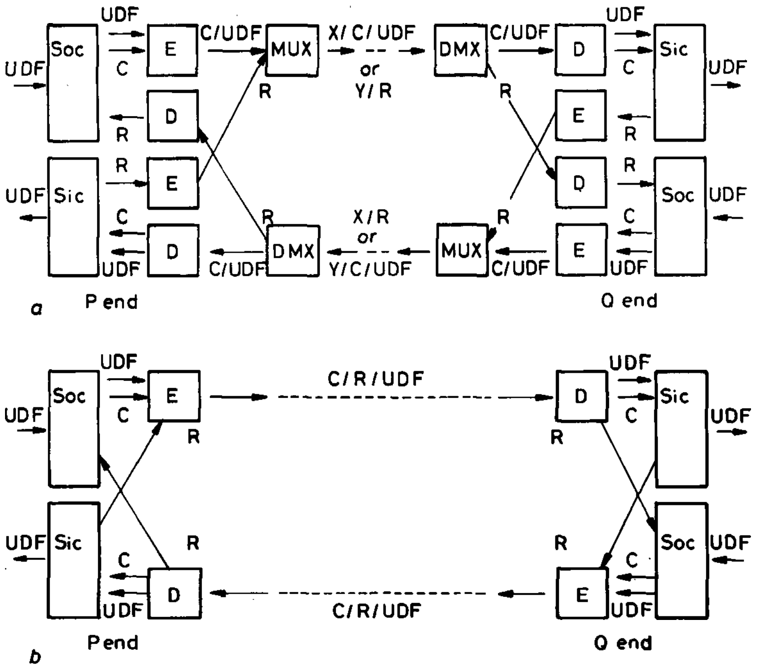


Fig. 5 (*a*) Independent simplex structure; (*b*) duplex structure

Key:	Soc	= source control entity	DMX	= demultiplexing entity
	Sic	= sink control entity	UDF	= user data field
	E	= frame encoding entity	C	= command field
	D	= frame decoding entity	R	= response field
	MUX	= multiplexing entity	X, Y	= address field values

configured back-to-back, each link being responsible for controlling the flow of user data in one direction only, or it may be designed as an integrated full duplex link. Figs. 5*a* and 5*b* illustrate the differences between these link control sublayer structures.

The essential distinction between these structures rests in the manner in which information is exchanged between corresponding source control (SoC) and remote sink control (SiC) entities so as to effect their operation in concert. In the independent simplex case the dialogue is effected using independently coded frames that contain information relating to the operation of a single SoC/SiC pair. This arrangement imposes the requirement for multiplexing/demultiplexing entities to enable sharing of the common medium between the SoC/SiC pairs. For the duplex case composite frames are used containing information relating to the operation of both SoC/SiC pairs, a technique sometimes referred to as response embedding or piggy-backing. This arrangement imposes the requirement for shared frame encoding/decoding functions.

The following table summarises the sets of frame formats needed to support the different link structures, one set for the independent simplex case and two equivalent sets for the duplex case, both of which are consistent with the duplex structure.

The essential point to note is that each of the format sets in Table 1 is self-consistent and complete. Each by itself is sufficient to support any end-to-end dialogue that may be defined to achieve the concerted operation of a source/sink entity pair. The attributes of self-consistency and completeness are fundamental. Before dealing with them some observations should be made concerning the attributes, from an implementation point of view, of the different link structures.

Table 1 Frame formats

System	Function	Format		
Independent simplex	send command with data P → Q	X/C/UDF		
	send command with data Q → P	Y/C/UDF		
	send response P → Q	Y/R		
	send response Q → P	X/R		
Duplex	send command with data	C/R_0/UDF}		{I_1/C/UDF
	send response	C_0/R}	or	{I_2/R
	send command and response with data	C/R/UDF}		{I_3/C/R/UDF

Key:

C	= command field of format
R	= response field of format
X, Y	= values encoded in the address field of format to differentiate the two streams
R_0	= value encoded in the response field to indicate the absence of a response
C_0	= value encoded in the command field to indicate the absence of a command
I_1	= format identifier code equivalent to R_0 indicating the presence of a command only
I_2	= format identifier code equivalent to C_0 indicating the presence of a response only
I_3	= format identifier code indicating the presence of a command and a response

5.1 Independent simplex structure

For this structure the MUX, DMX, SoC and SiC functions may all be implemented as independent functions in exactly the same way as combinations of these functions can be implemented independently within the wider structures considered at earlier stages.

5.2 Duplex structure

For this structure an encoding entity must have access to the internal states of its related local SoC and SiC entities for the purpose of generating composite formats. Equally a decoding entity must be associated with a scheduling mechanism of some kind of schedule the activities of its associated local SoC and SiC entities in the event of a composite frame being received.

It would appear, therefore, that in spite of the complete logical separation made between command and response fields and the independence given to the encoding of these fields within the idealised formats identified in Table 1, the duplex system nevertheless implies a higher degree of interdependence at the implementation level of the functional entities comprising a link protocol machine than that obtaining for the independent simplex case.

It may also be seen that the apparent simplicity of the duplex structure relative to that of the simplex is a deception. The encoding/decoding entities of the duplex structure take on exactly the characteristics ascribed to the message transformation functions considered at stage 1. They become key elements binding the structure.

The general conclusion is that if link control information relating to laterally independent entities is combined in a common format so too will these entities be likely to be integrated within a monolithic implementation.

6 Defects of standardised HDLC formats

We now return to the attributes of self consistency and completeness of the sets of formats given in Table 1. If an address and control field encoding regime is adopted that is not complete, then its missing elements must be made good in some way. This can be done either by adopting the equivalent element from some other regime or by defining additional end-to-end dialogue and procedural rules. Either way unnecessary complexity is necessarily generated. If foreign elements are used, then the protocol machine structure will be distorted by becoming a hybrid of the structures supported by its native and foreign elements. If additional procedural rules are defined, then the structure is again distorted by the inclusion in it of the functional entities needed to implement the rules.

This is precisely what may be seen to have happened in the development of Balanced Mode HLDC procedures. The control field structure of the HDLC format is basically that required to support the duplex link structure of Fig. 5*b* in that it makes provision for logically separate command and response subfield coding. However, the control field includes a single-bit subfield that is sometimes required to be seen as a part of the command subfield and sometimes as a part of the response subfield. This bit is the Poll/Final bit having the meaning Poll in a command and Final in a response. A duplex link structure demands the definition of two bits in the control field to permit Poll and Final bits to be transmitted simultaneously in a composite command/ response frame. Because no such facility is provided, some other means had to be devised to achieve the effect of two bits.

The means chosen was to use address field values, as for the independent simplex structure, to differentiate commands and responses and thus qualify the meaning of the single bit. The inevitable consequence was conversion of the formats into a set naturally supporting a highly redundant dual duplex protocol machine structure. This led in turn to the need to define unnecessarily complicated procedural rules to make some kind of sense of the redundancy, thereby compounding the original error and ensuring that nothing other than a monolithic protocol machine structure would suffice to meet the requirements of the procedures.

The error also had the effect of defeating the aims of duplex encoding of the control field in that for some link control functions reversion to simplex coding has to be resorted to.

7 Implementation experience

The problem for the designer of an HDLC protocol machine is that of deciding how best to achieve some of the modularity and relative simplicity of the independent simplex structure of Fig. 5*a* while at the same time achieving conformance to HDLC standards.

This was achieved in ICL by specifying a two-sublayer implementation structure. The upper sublayer was defined as incorporating all but the MUX/DMX elements of the Fig. 5*a* independent simplex structure, the lower sublayer being made up of functional elements performing the combined functions of the MUX/DMX elements of the Fig. 5*a* structure and the composite frame encoding/decoding elements of the Fig. 5*b* duplex structure.

Idealised and much simplified procedures were defined for the upper sublayer. Lower sublayer format and protocol conversion procedures were then defined to support the upper sublayer procedures. These also turned out to be surprisingly simple.

This specification technique, while significantly aiding in the design of a conforming HDLC protocol machine, did not, of course, yield a truly modular structure exhibiting the properties of lateral and layer independence for all the reasons previously given.

Theoretical work of a similar kind to that given in this paper has also been exploited in ICL implementations of CCITT rec. X.25[6].

8 Conclusion

The foregoing has attempted to show that the structure and properties of a system are very dependent on the structure (syntax) of the information required to be generated and processed by the system, and not so dependent on the meaning (semantics) of the information, as might intuitively be thought to be of more significance.

It has been shown that system structure and message structure must be seen as two faces of the same coin; that if they are not so regarded, then, in the final analysis, message structure will impress itself on system structure with consequent magnification of system cost and loss of layer and lateral independence.

Ideally, the message header structures required to support a layer protocol should not be defined until a definition has been given of the structure of the protocol machine envisaged as being needed to implement the protocol. They should then be defined in strict conformity with the intended protocol machine structure.

It is believed that adoption of this methodology in the development of OSI layer 1–4 protocols generally would significantly assist in reducing to a minimum the volume of English prose needed in the first instance to define the protocols and, through such reduction, subsequently assist in their formal description[7].

Acknowledgments

Thanks are due to Dr. Ken Turner for his encouragement to write this paper and his subsequent assistance in the editing of the text to make it readable.

References

1 BRENNER, J.B.: 'IPA networking architecture', *ICL Tech. J.*, 1983, **3** (3), 234–249.
2 TURNER, K.J.: 'The IPA telecommunications function', *ICL Tech. J.*, 1983, **3** (3), 265–277.
3 ISO IS 7498 Information processing systems – open systems interconnection – basic reference model.
4 ISO IS 7809 High level data link control procedures – balanced mode.
5 ISO IS 3309 High level data link control procedures – frame format.
6 TURNER, K.J.: 'Designing for the X.25 telecommunications standard', *ICL Tech. J.*, 1981, **2** (4), 340–364.
7 TURNER, K.J.: 'Towards better specifications', *ICL Tech. J.*, 1984, **4**, (1), 33–49.

Correspondence

A suggested extension of ICL DAP parallelism

This note suggests a way in which use could be made of dual-port RAM to extend the parallelism of the ICL DAP, making it possible to pipeline its already parallel operations.

From R.M.R. Page, School of Information Sciences, Portsmouth Polytechnic: Array processors are normally considered to belong to one of two types: multi-processor or pipeline/vector array processors.

The advent of true dual port RAM, i.e. RAM that can be accessed simultaneously by two processors, makes the extension of the ICL DAP to a multi-processor pipeline array system more attractive.

The proposed system is indicated in Fig. 1. Each horizontal plane of the three-dimensional array comprises a DAP as currently implemented, except that dual port RAM is used so that the PEs (processor elements) from the $(i + 1)$th plane can receive as input the output from the ith plane above. The spare port on the final plane would be used for output. Thus when the array is fully primed, each plane is simultaneously processing one stage of the pipeline.

The number of planes or stages required would depend on the function being performed, and on the amount of work performed by one stage. To this extent, the amount of hardware expansion would be flexible. One stage could execute a number of machine instructions, thereby keeping the hardware expansion to a minimum. Alternatively a stage might carry out an operation on one-bit operands, working at the primitive control level. This would provide an extremely fast throughput. The whole array would be synchronised to the slowest stage, and input/output would probably be the deciding factor. If pipelined matrix operations were envisaged, then specialised I/O hardware would be required to achieve maximum performance.

Eric Baddiley, ICL PERQ Business Centre, Kidsgrove, Staffordshire. My comments on the proposal by R. M. R. Page to make a pipelined DAP. What is actually proposed are multiple DAPs, since each PE plane in the diagram requires its own MCU, code store etc. This would obviously be a BIG machine.

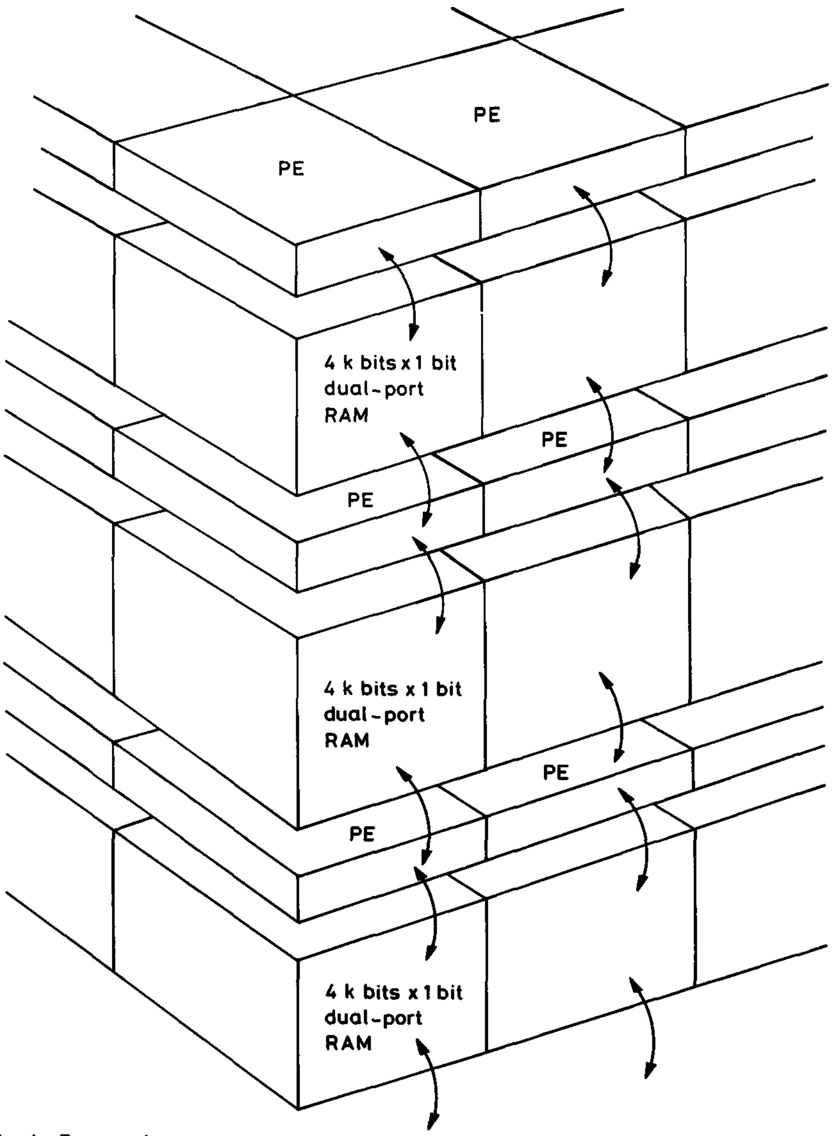


Fig. 1 Proposed system

Since for any given level of chip integration a dual port RAM will be smaller in bit capacity due to the extra logic required (4 K bits × 1 dual port vs. 16 K bits × 4 fast static RAMs, a factor of 16 smaller) then the number of RAM chips required in a machine for a given total capacity will be that much larger. Two port RAMs are also physically bigger since two sets of addresses and control lines are required per chip. The cost of these special RAM chips also tends to be at a premium compared with the industry standard, multiple-sourced devices.

It is suggested that the array be expandable to suit the problem to be solved. This would require 2048 connections just for the data bits alone, that is 1024 connections to the array above and below this one. This is hardly practical and exaggerates the well known problem with the DAP architecture – lots of connections to memory (it is also its strength).

Even if the array is not expandable the number of interconnections is vast. Unless the whole array is built on one board (this is very unlikely) then the number of interconnections would be unrealistic. The densest connectors I know of provide 48 connections per inch, and ignoring power and ground the data pins alone would require over 50 inches of solid connector length. The problem is twice that of expanding the memory of the current DAPs.

Some solutions to this may be to only make a 16 × 16 DAP pipelined and have connections inside the PCBs and mate the boards sideways (Fig. 2).

Since the only path from any store chip to the next is through a PE array it seems likely that the PEs will spend time doing nothing more than moving data, although this will depend upon the algorithms in use.

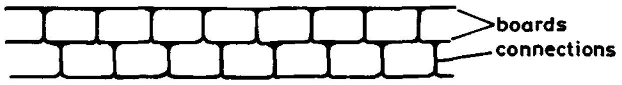


Fig. 2 Sideways connection

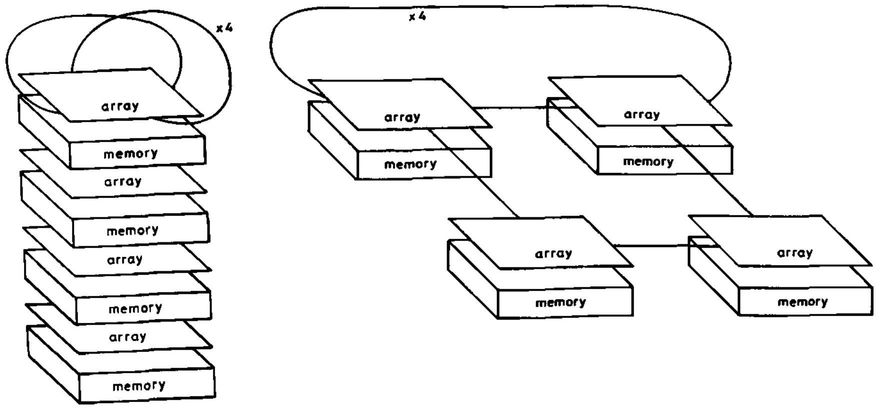


Fig. 3 Architecture using reconfigurable arrays

The suggestion is that the DAP be physically configured for the particular application; this detracts from the generality of the DAP. In particular, if an application did not require a pipeline architecture but instead a single PE array and a large contiguous address space this design would be useless since there is no direct access from one PE array to all the RAM in the machine. An alternative view to this would be to consider the array as reconfigurable so that a 4 level pipeline could be reconfigured to make the DAP PE array four times the size. The problems of multiplexing the edge connections seems to require a lot of hardware and interconnections (Fig. 3).

If a dedicated pipeline DAP architecture IS required then a few alternatives come to mind.

Since data only flows one way through the pipline there is no need for dual port RAMs; conventional RAMs can be used with an efficient implementation as shown in Fig. 4.

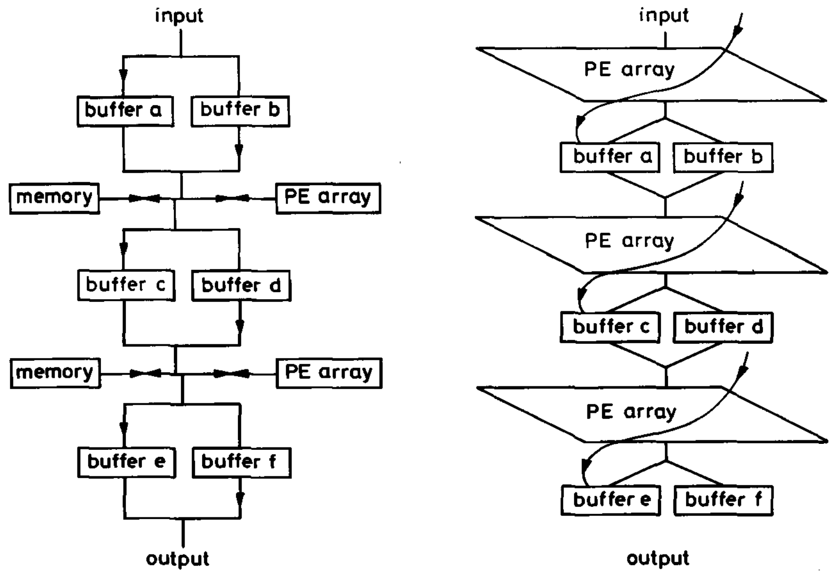


Fig. 4 Use of buffers with conventional RAMs

Buffers a, c and e are written to and b, d and f are read from. On the pipeline beat these are swapped over implementing a double buffer system. By using buffer memories with separate input and output pins no extra hardware is required, simply gating the output enables and write enables. Each PE array requires mass working store since the buffers only contain the data being processed. Although this appears to take more store chips, because 4 bit wide chips can be used, less chips are required, they are also much more dense and are standard, cheaper parts.

Another method would be to use video RAM chips with built in shift registers (Fig. 5). The data could be shifted from one set of PEs memory into the next while the arrays were processing the other buffer. Again a double buffered system but with both buffers in the one chip. Because the shift registers are big (>256 bits) a memory cycle is stolen by the shift registers infrequently leaving most of the time for the PEs. This seems to be better than the first method except that all video S/R (shift register) RAMs I know of are slow dynamics.

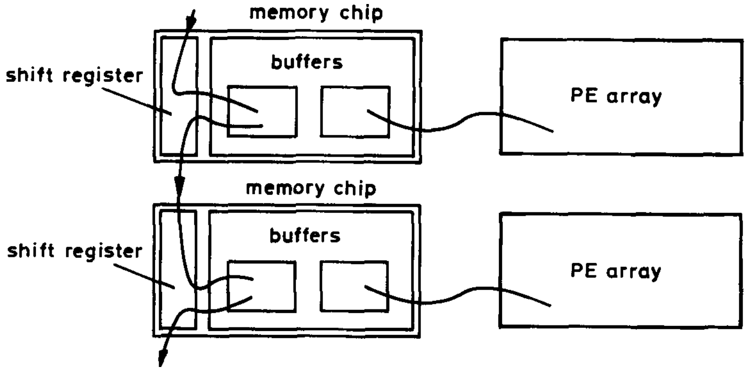


Fig. 5 Use of video RAM chips with built-in shift registers

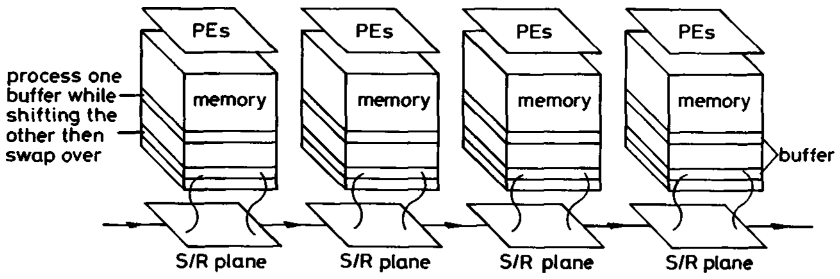


Fig. 6 Pipelining the existing DAPs

Another alternative is to pipeline the existing DAPs using the S/R plane to shift the data while the PEs process the other buffer (Fig. 6). This will only be efficient if the shifting of the buffers is complete before processing the other buffer is complete. The S/R can shift at an average of 100 nS per 32 bit word or 3·2 uS per DAP plane. This requires an average of 21 DAP cycles per plane of data before the PEs will be held up waiting for the S/R plane transfers to finish.

Notes on the authors

D.J. Ackerman Message structure as a determinant of message processing system structure

David Ackerman joined ICL (then ICT) in 1958 after serving 14 years in the Radio/Electrical branch of the Royal Navy. In 1966, following completion of development work on the 1900 Series standard peripheral interface, he joined the newly formed Data Systems branch to work on the 7000 Series communications front-end processors for the 1900 and 2900 Series machines. In 1970 he served as a member of the ICL team set up under contract to the then GPO to study the requirements of a UK packet-switching network. During this study he formulated what became known as the 'onion ring' method of communication systems specification and design, based on the principle of protocol layering. He was subsequently active with others in promoting adoption by ISO of this principle in the definition of a reference model for Open System Interconnection. From 1978 onwards he has remained an active contributor through BSI and ECMA to the work of ISO in the areas of ISO Layers 1–4 architecture and standards and has been primarily concerned with the production of standards for the ICL Network Product Line/Information Processing Architecture, in line with those of ISO.

M. Campbell-Kelly ICL company research and development. Part I 1904–1959

Martin Campbell-Kelly graduated from the University of Manchester with a B.Sc. in Computer Science in 1968 and received a Ph.D. in History of Science in 1980. He is now deputy chairman of the Department of Computer Science in the University of Warwick. He is presently engaged on a number of computer history projects, including a corporate history for ICL. He is editor of the MIT Press Reprint Series for the History of Computing and the Collected Works of Charles Babbage, and also an editor of the *Annals of the History of Computing*.

D.A. Duce Formal specification – a simple example

Dave Duce has a Ph.D. from the University of Nottingham. He joined the Science & Engineering Research Council's Rutherford Appleton Laboratory in 1974 and has been involved in computer graphics since 1975. He was involved in the development of the Graphics Kernel System (GKS) being one of the editors of the International Standard, and is co-author of a book on GKS. He has worked with Elizabeth Fielding on a joint research project for the specification of GKS since 1984.

J.P. Fellows (as B.A. Kitchenham)

John Fellows graduated from the North Staffordshire Polytechnic with an Hons. degree in Computing Science, having spent a year at ICL Kidsgrove working on software metrics as part of his degree course. He is now working for AB Food where he is involved in statistical control.

E.V.C. Fielding (as Duce)

Liz Fielding has an M.Sc. in Computation from Oxford, during the course of which she was introduced to formal specification and became interested in this, She joined the Rutherford Appleton Laboratory in 1981 and is currently working with David Duce on a research project for the specification of the Graphics Kernel System, GKS.

M.D. Godfrey Innovation in Computational Architecture and Design

Michael Godfrey received a B.Sc. degree in Engineering from the California Institute of Technology in 1959 and a Ph.D. in Mathematical Economics from the London School of Economics and Political Science in 1962. He then joined the faculty of Princeton University, first in Economics and later in Economics and Statistics. During this time he was on the research staff of the Econometric Research Program. In 1967 he became associated with Bell Telephone Labs in Murray Hill, N.J. This association with Bell Labs and the Corporate Planning Organization of AT&T continued until 1977. In 1969 he joined the Statistics Section in the Mathematics Department at Imperial College of Science and Technology. He held this position until 1977 when he joined Sperry Computer Systems, where he was Director of Research until 1983. His current position with ICL is Head of Research.

P. Henderson The *me too* method of software design

Peter Henderson is Professor of Information Technology at the University of Stirling, Scotland. Previously he has held appointments at Oxford University and at the University of Newcastle upon Tyne in England. He is the author of a popular textbook on functional programming.

M.J.J. Holt Recent developments in image data compression for digital facsimile

Murray Holt obtained a B.Sc. in Mathematics at Manchester, and in 1978 was awarded a Ph.D. in Physical Chemistry for work on the numerical analysis of chemical reaction data at Portsmouth Polytechnic. He then worked as a programmer/analyst, specialising in statistics and graphics, for small companies in Lancaster and Loughborough. In 1981 he joined Loughborough University's Engineering Maths department, researching the application of computer graphics in the statistical analysis of industrial processes. Dr. Holt is currently engaged on an ICL-sponsored research fellowship in bi-level image compression, in the Department of Electronic and Electrical Engineering at Loughborough University.

A.P. Kitchenham (as B.A. Kitchenham)

Allen Kitchenham received a first class Hons. degree in Mathematics and Computing and later a Ph.D. in Computer Science at the University of Leeds. Since leaving university he has worked as a designer and implementer of ICL's VME operating system in the area of catalogue and file management. He is interested in techniques that assist the production of high quality code and has been involved in pilot studies of data collection and Fagan inspection. He has been involved also with the introduction of Inspections into the VME production groups.

B.A. Kitchenham The effects of inspections on software quality and productivity

Barbara Kitchenham received a B.Sc. in Mathematics and Statistics (1969), M.Sc. in Statistics (1970) and Ph.D. (1972) at the University of Leeds, and worked for three years as a statistician before joining ICL as a systems programmer. After several years working on the production of the VME operating system she moved into the area of software metrics and since 1980 has worked on the problems involved in measuring software and in integrating measurements into the development process. She has published a number of papers on software development, based on empirical studies of VME production. She is now working for STL in connection with Alvey and Esprit projects.

B.G.T. Lowden REMIT: a natural language paraphraser for relational query expressions

Barry Lowden graduated from King's College, London in 1964 with a first degree in Physics. He then joined the Plessey Company where he worked in Computing for eight years, being finally responsible for all systems development at the Ilford site. In 1972 he studied for an M.Sc. in Computer Science at London which he gained with distinction. Since then he has been a lecturer in Computer Science at the University of Essex. His main research interests lie in the areas of information retrieval, relational databases and query languages. For the last two years he has been funded by the I.C.L. University Research Council to carry out work on natural language front ends to relational systems. The research team at Essex is currently developing a front end to support queries both on and about the data held in the database.

C. Minkowitz (as Henderson)

Cydney Minkowitz has an MSc in Computation from Oxford. She joined ICL in 1982 working on knowledge engineering in The System Strategy Centre at Stevenage. In 1984 she transferred to the Software Engineering Technology Centre at Kidsgrove and is currently on secondment to the Department of Computing Science at the University of Stirling working on a collaborative Alvey project.

A.N. de Roek (as Lowden)

Anne de Roek graduated with distinction from the University of Leuven, Belgium, with a degree in Linguistics, English and Dutch, in 1979. After a year with the Belgium Telephone Company, working in the field of library automation, she was attached to ISSCO, a research institute of the University of Geneva, Switzerland, where she participated in the development of a number of machine translation systems including the EEC-sponsored EUROTRA project. In 1983 she took a M.Sc. in computer Studies at the University of Essex. She has remained at Essex as a Senior Research Officer and has participated so far in two ICL contracts in the area of natural language front ends to databases.

V. West Natural language database enquiry

Vincent West graduated from Gonville and Cauis College, Cambridge, with a B.A. in Mathematics in 1964 and joined the Central Electricity Generating Board as a programmer. He moved to ICL in 1966 to work on Fortran compilers, including the design of code-optimisation methods. After a variety of software design work, including operating systems, he specialised in databases. This has included work on the Codasyl system IDMS, relational databases including the ICL Personal Database System and Querymaster, and most recently natural language database interfaces.

C.S. Xydeas (as Holt)

Costas S. Xydeas received the first degree in electronic engineering from Vranas Higher School of Electronics, Athens, Greece, in 1972 and the M.Sc. and Ph.D. degrees in electrical engineering from Loughborough University of Technology, Loughborough, England, in 1974 and 1978, respectively. In 1977 he joined, as a Research Fellow, the Department of Electronic and Electrical Engineering of Loughborough University and has worked on low-bit-rate coding of speech signals. He is at present a Senior Lecturer at Loughborough teaching telecommunications and signal processing. He is also directing the M.Sc. course in digital communication systems and a research group in digital coding and processing of speech and visual signals.