NEWSLETTER

EUUG

European UNIX® systems User Group

Volume 6, No. 3
WINTER 1986

CONTENTS

EUROPEAN UNIX SYSTEMS USER GROUP NEWSLETTER

Volume 6
Number 3

This newsletter was set in Times Roman on a LaserGrafix LG800 printer, driven from the UNIX time-sharing system.

Thanks to:

Jean Wood of DEC	Content
Alain Williams of Parliament Hill Computers	Content
Sally Rutter of The Instruction Set	Collation and Typesetting

So how is the EUUG funded?

Nigel Martin

EUUG Treasurer

1. HISTORY

The EUUG was formed approximately 10 years ago in 1976. A small group of academics started the group to further the interests of UNIX and to allow them to meet on a fairly regular basis to discuss in-depth topics to do with the UNIX operating system. Because it was a small group, nobody cared about money. No money was really needed for anything. Most of the people attending meetings and committee meetings footed the expenses out of their own pockets. There was no infrastructure to support.

There were approximately two conferences per annum; each being run on a university site. One of the committee members would typically organise the conference. Any risk would be underwritten by the University department for whom that person worked. Conferences tended to make a small profit. Activities were all based around the UK with a small number of contributions from other parts of Europe.

2. FORMATION OF EUUG

As time passed, interest from other countries, in particular The Netherlands, suggested a change in structure to accommodate individual national needs.

It was decided to spilt the EUUG into a number of national groups. Before the split, every EUUG member paid a subscription to the group to enable it to continue to operate. After the formation of national groups, it was clear that the EUUG still needed a source of income; so too did the national groups. It was clear too that national groups needed the largest proportion of their income to fund local activities, but the EUUG needed sufficient to fund all the centrally provided services.

The EUUG had built up some central reserves, over many years, from the profit made at each conference. It was decided to retain this fund as a buffer to assist with the financing of major European events.

A set of rules of operation were constructed to govern the amount of money to be paid by a national group into the central funds. At the time of formulation of these rules, there were no visible financial problems with any of the groups. It was the *intention* that national groups would pay 40% of their *total* income over to the EUUG, with a minimum of £20 per member.

Unfortunately, there were a number of errors in the written word contained within the original by-laws. This resulted in certain national groups not actually operating by *the spirit* of the operation, but by what was stated. This situation was unfortunate and hence the committee were obliged to rectify it.

In the early days, certain national groups were offering an indirect subsidy to others. Whereas this is fine as a short-term solution, it cannot operate permanently. Hence, as time progressed, and as the European Community grew larger, it was necessary to straighten out the funding of each national group.

3. *FINANCIAL REVIEW*

Being European costs money. This is unfortunately a very true statement, which EUUG has learned to its cost. Running European conferences is more expensive than running small local ones. The risks attached to each are greater. As the success of the group increased, so the need to run bigger (and better) conferences became more apparent. Associated with large conferences are large risks; possibly large profits too.

The committee, having studied the finances, decided it was most unwise to rely on substantial income from each conference to fund the essential activities of the EUUG. Therefore, some time was spent studying the accounts in great detail. The EUUG, at the time, had an annual turnover in excess of £100,000 but a subscription income of less than £13,000. This is clearly an unacceptable way to operate a user group, in which each of the members is in some way liable for any loss the group encounters.

It is most unrealistic for members to be liable for a loss over which they have no control: for that reason they appoint the executive committee. The executive committee would not be doing their job correctly if they were to allow a group to run into any financial difficulty and subject members to any unnecessary risk.

Since the subscription income was extremely low, and the rate had not been changed for many years, it was agreed to increase this. At the same time, there was a strong desire to remove the financial anomalies which were present for historical reasons. Some groups were being very heavily subsidised. This did not appear to be a fair or workable structure for the future. Indeed, in certain cases it was not advantageous for the EUUG to accept new groups because it would substantially reduce their total working funds.

4. *A NEW STRUCTURE*

It was clear that a new structure was required. After considerable thought, regarding the relative merits of charging individual and institutional members at different rates, it was decided that since each type of member cost EUUG precisely the same amount of money, the fair way was to ask for a per name contribution from a national group, regardless of the type of member.

But, how much is reasonable? The calculation performed was extremely simple: take the cost of running the EUUG after conferences were extracted, and divide the result by the number of registered members. Clearly a small reserve was added to cater for the potential inadequacies of a new budgeting system. The magic number was £40 per member per annum.

The committee have regularly been asked to produce cheaper conferences. With the new fee structure in place it was now seen as a possibility to allow the conferences to break even and not make a profit.

The new structure also provided a sound firm basis for the EUUG to operate — we now have budgets.

4.1 Copenhagen

Unfortunately, it was necessary to introduce this scheme rather urgently since the magnitude of the problem had not previously been understood. It was felt too dangerous for the group to be allowed to continue in the old manner for another half year. Hence the proposal was put to the Governing Board at the Copenhagen meeting to suggest a new level of contribution from the national groups. In some cases the new system appeared to have little effect on the groups: a group charging £100 per annum supplying 40% of their income would indeed pay approximately £40 to the EUUG. It was assumed by the committee that £100 per annum was not an unreasonable sum to charge, and that £60 per annum was not an unreasonable amount for a national group to retain for its own purposes.

Many of the national groups were certainly unaware of the financial situation that prevailed for the EUUG. After much discussion, all persons at the governing board agreed that the new financial structure was essential to the well-running and hence the future of the EUUG. To that end, the motion was passed and national groups were asked to pay £40 per name on the address list starting at the end of 1985.

4.2 Present State

We now have a workable structure where the income is directly proportional to the expenditure. For each additional name added to the address list, there is an additional amount of income and a corresponding amount of expenditure on newsletters, postage etc. This is a fair system. No group is treated differently from any other. No-one is subsidised.

Although each national group had some difficulty in the beginning, all national groups have agreed to pay the new fees. Most national groups have already done so.

4.3 Financial Structure of the National Groups

Although there is a flat fee structure at the UEUG level, it was always assumed that national groups would operate a multi-tier system. The climate in each country is undoubtly different and will always remain different. For that reason it was assumed that national groups would take their total outgoings to the EUUG, and divide this amount appropriately amongst their membership. For instance, it seems reasonable to charge commercial users considerably more than individual members. Some groups may choose to subsidise student members and allow them to join for considerably less than the £40 per annum that is paid to the EUUG. By making the problem a national problem it gives each group infinitely more flexibility and allows each group to structure their fees according to national interests.

National groups in most cases supply services in addition to those offered by the EUUG. The extent to which this is true can be reflected in the fees charged by that national group. It is assumed that national groups who do a lot would charge considerably more than the EUUG's rate; while national groups who do little or nothing would charge approximately the same rate as the EUUG, with a small levy to cover administrative costs of their own.

Some national groups elect to organise local events: these are outside the funding structure discussed above. All profit from such events would be retained by the national group.

5. EXPENDITURE

So where does the money go? Broadly speaking the expenditure of the EUUG can broken up into five categories.

The first is for central administration in the form of support for Owles Hall: this occupies approximately 50% of the group's income.

The second is the executive committee which requires approximately 15% of the group's income. Included in this category is the cost of running the committee, the cost of sending members to meetings and the cost of setting up and organising these meetings. There is little that can be done to reduce the absolute cost of these meetings. They are currently arranged for the location which involves minimum travel expenditure.

The third category is the newsletter. This currently takes approximately 20% of the group's resources but it is expected that it will take more in the future since the standard of reproduction of the newsletter is rapidly being increased.

The last 15% of the group's income is spread across numerous other activities including relationships with /usr/group/UK, relationships with UNIX Europe, funds for Internationalisation etc.

All funds which exist in the EUUG are used as a buffer to support conferences. Running a large conference usually means paying out a considerable amount of money in advance. It also means underwriting any potential losses. We still have a requirement to run large conferences, but the larger the conference the greater the potential loss and hence the greater the buffer needed in order to underwrite said loss. We have found in the past that many conference halls will not accept bookings unless there is some evidence that we are actually able to pay; or at least we have forwarded a substantial deposit. Any income from a conference will simply go to enhance this buffer.

Now that the EUUG is no longer reliant on the income from conferences, it is in position to offer and attractive revenue split between itself and the organising national group. This allows a national group the opportunity to significantly supplement its income.

The last conference, held in Manchester, was the first to benefit from the fact that income from conferences is no longer essential: it was cheap!

6. *CONCLUSIONS*

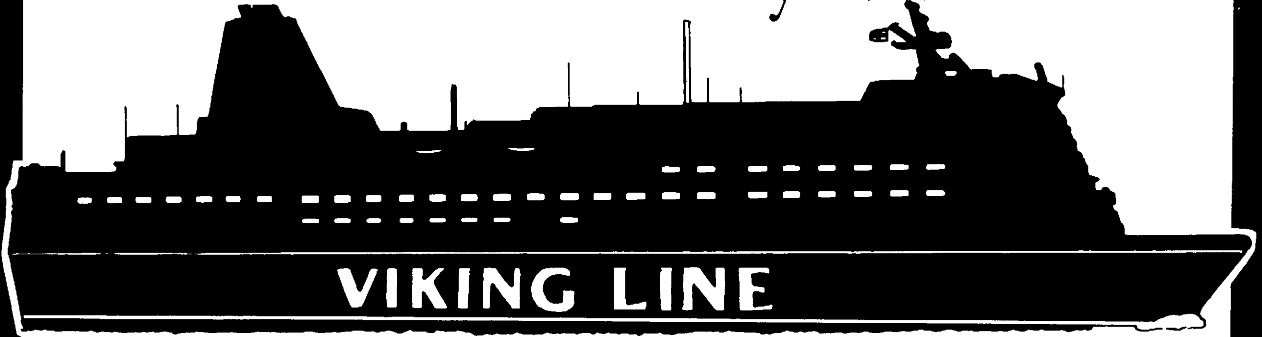
The EUUG is now financially sound. The new fee structure is fair and flexible. All that remains is to answer the question "is £40 per annum a reasonable contribution to expect for each name on the membership list?" I believe the answer is yes, particularly when we consider that this is less than the individual saving for each member that attends one of the European conferences.

EUUG

European UNIX® systems User Group

"UNIX® GROWS UP"

A major 5-day event in
Finland and Sweden
on board the M/S Mariella
11th – 15th May 1987



The Event

The European UNIX systems User Group is to hold one of its most important events yet — in May 1987.

Its **Spring 1987 Conference and Exhibition** will be held over three days (12th – 14th May) on board the Viking Line's luxury passenger ship — the M/S Mariella — with **Tutorial Sessions** at the Hotel Dipoli Conference Centre in Helsinki on 11th and 15th May.

The **Conference and Exhibition** will be staged as the M/S Mariella sails between Helsinki and Stockholm.

The Programme

Under the broad heading **UNIX® Grows Up,** the Conference will encompass a variety of both new and old developments in the UNIX field and will also look into the future through the eyes of such leading specialists as:

Bill Joy — Sun Microsystems Inc
Rob Pike — AT&T Bell Laboratories
Brian Redman — Bell Communications Research
Bjarne Stroustrup — AT&T Bell Laboratories
Andrew Tanenbaum — Vrije Universiteit, The Netherlands

Nine different UNIX topics will be presented during the Tutorial Sessions

For further details on this major UNIX event, contact:
The Secretariat: European UNIX® systems User Group, Owles Hall, Buntingford, Herts SG9 9PL, UK. Tel: Royston +44 (0) 763 73039 Facs: Royston +44 (0) 763 73255
Network address: euug@inset.uucp

®UNIX is a Registered Trade Mark of AT&T in the USA and other Countries

AFQL - A Flexible, General Purpose Interface to Relational Database Management Systems under UNIX

Paul Clark
Andrew Simms

Data Logic Limited

1. INTRODUCTION

Relational database management systems are becoming widely used commercially, and increasingly so within the UNIX community. This paper describes a flexible, general purpose interface for handling the execution of relational database retrievals and updates. It has been used in an implementation of an IBM compatible SQL under UNIX, and is applicable to other non-procedural languages.

The interface comprises two major components. The first, the Relation Manager, contains a full set of relational database functions providing facilities for the creation and maintenance of relations, tuples, attributes and indices. A range of data types is supported, together with an implicit locking and transaction management strategy. An interface to this component is available to application programmers.

The second component is the Amplified Functional Query Language (AFQL), a language in which the database is viewed as functions mapping between relations, tuples and attributes. It allows retrievals and updates of arbitrary complexity to be constructed, including recursive constructs, whilst minimising storage requirements and employing lazy evaluation techniques. Examples of its use are illustrated, together with UNIX- and C-related features of its implementation.

The UNIX environment provides tools which greatly assist in the support and maintenance of structured query languages built upon the AFQL interface.

Let us set the scene with a forty second summary of forty years of computing.

To begin with, users accessed and stored their data in a totally unstructured manner. The advent of hierarchic and networked databases provided a more structured route for users to reach their data, via procedural applications interfacing to a database management system (DBMS).

Whilst this approach is well proven, and gives many users a warm, comfortable feeling, it is essentially inflexible. A contemporary solution to this inflexibility has arrived with the use of relational database management systems (RDBMS).

The use of an RDBMS, with a non-procedural query language (such as SQL), is beginning to supersede the traditional procedural approach for systems design and implementation (figure 1). SQL is now the de facto standard query language used to interface to RDBMSs, and has in fact been proposed as the ANSI standard.

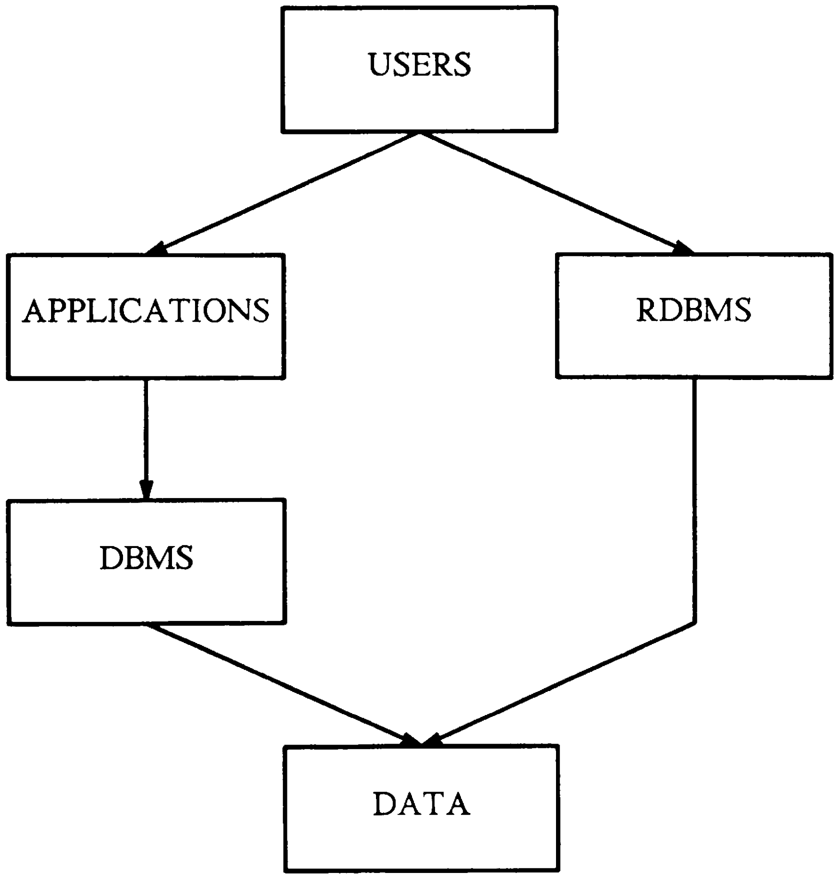


Figure 1. RDBMS

Now you may be thinking that you have heard all of this before! And so you may have, but you probably think that you have to use SQL. We believe that SQL should be just one way of using a relational database.

Over the next few years there will a variety of different man-machine interfaces utilising RDBMSs, and SQL is likely to be relegated in importance. If SQL and the RDBMS are tightly bound together, it will be that much harder to implement these new systems.

The implementation techniques we have used within our RDBMS are innovative, enabling several diverse man-machine interfaces to be supported concurrently.

We describe two general purpose and powerful components, that can be used to implement either procedural or non-procedural applications, using RDBMS.

The procedural level interface is provided by the Relation Manager (RM), while a functional query language approach (AFQL - Amplified Functional Query Language) has been taken for the non-procedural interface. This is shown schematically in figure 2.

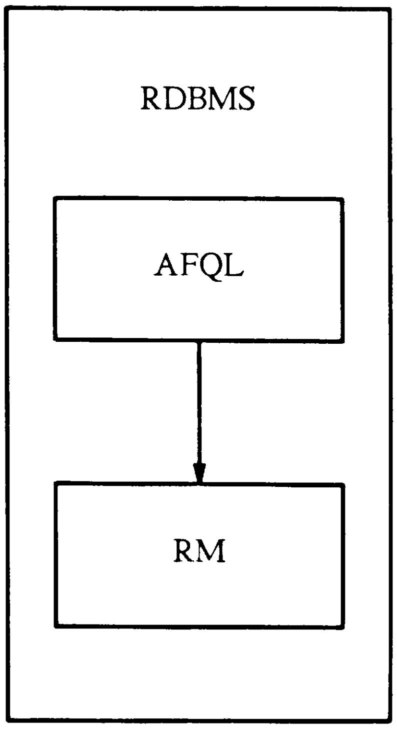


Figure 2. RDBMS Components

We will go on to describe the functionality of each of these components in some detail, before concluding with some illustrations of their use.

(An appendix provides a formal definition of AFQL constructions used to illustrate this paper.)

2. *RM FEATURES*

A general purpose RDBMS needs flexible facilities for manipulating and managing the individual components which comprise the database. Within our implementation these facilities are provided by the Relation Manager (RM).

For the RM to be flexible enough to support a general purpose RDBMS it must provide (at least) the following facilities :

- A Comprehensive Database Access Capability

  An RDBMS will need to support not only extensive ad-hoc retrieval access, but also full capabilities for data modification. If RDBMS systems are to be successful in the long term then they must provide the full range of facilities currently available from conventionally organised, specialist databases.

- A Comprehensive Dynamic Data Definition Language Capability

  Facilities for the dynamic creation and manipulation of the data structures that make up the database must be provided. There is a need for relations to be created and deleted, indices to be allocated and de-allocated, and new attributes added and removed.

- Data Dictionary Support

  An essential requirement of an RDBMS is a data dictionary. This dictionary should be readily available to the user and support definitions and descriptions of all data attributes, stored and perceived tuples, database names, indexing, user help, macro and routine definitions. The maintenance of such a dictionary must be automatic, and always reflect the current status of the database.

- Access Control

  RDBMSs provide a large range of facilities and support multiple applications for a large range of users. Maintaining such a database relies heavily on having a comprehensive access control capability, providing varying levels of access to both user and system data.

- Transaction Management

  The essential requirement from any database management system is that the user knows that his/her data is safe at all times. Not only must the safety of the data be protected from system failure, but also (as far as possible) from inadvertent user failure.

- Locking

  Databases supporting multi-user update access must also support a locking mechanism which resolves update clashes. Misunderstanding the use of the locking strategy can lead to problems with the design and implementation of applications. To prevent misuse the locking strategy must be comprehensive, easy to use and be able to recover from deadlocks.

- Database Integrity

  An RDBMS supports both user data and system data. System data must be made available to users as required (for example, to provide catalogue facilities). This makes the integrity of the system data of paramount importance, a problem complicated by the fact that some users may have update access to it.

- High Performance

  The traditional hierarchic/networked approach uses structures which are inherently optimised for high performance. On the other hand, an RDBMS is manipulated using formal set processing techniques, which may involve large amounts of processing of both system and user data. Although users may be willing to trade some performance for extra flexibility, overall the RDBMS performance must be somewhat comparable to that provided by current DBMSs.

- A Distributed Database Capability

  The architecture of the RM component must be such that, if a particular implementation of the UNIX environment provides an efficient inter-processor communication capability, it will be amenable to enhancements incorporating support for distribution of the Relational Database across several processors.

- A User Application Program Interface

  Whilst the RM may be transparent to users owing to the higher level interfaces that are built upon it, certain applications may have a requirement to use the specialist, low-level, high performance interfaces direct to the RM. Therefore these interfaces should be made available to high privilege, specialist users.

3. *RM FUNCTIONS*

The RM provides some of the above facilities via the following set of functions:

- Find first/next tuple in relation

  Simple predicate matching within a relation can be performed by the RM, allowing the use of performance aids such as multi-key retrieval, and index optimisation to be performed automatically. These functions allow all matching tuples in a relation to be retrieved.

- Insert, Rewrite and Delete tuple in relation

  At any position in a relation (located via Find tuple) updates may be performed.

  During update operations all necessary relation and tuple locking is performed automatically by the RM. In addition there are facilities which allow entire relations to be explicitly locked by a user.

- Create/Delete Database

  The creation of a database automatically creates and initialises all necessary system relations.

- Create/Delete Relation

  The creation/deletion of a specific relation automatically causes the RM to modify all relevant system relations, thus ensuring database integrity.

- Create/Delete Index

  The creation/deletion of indices is performed by the RM, allowing all users (whether using a high level query language interface or a low level, direct RM interface) to benefit (or suffer) immediately from the database modification.

- Access Control

  User permissions are presented as tuples within system relations, and may be modified using the standard relation modification facilities. However, for performance reasons, the access permissions are stored in bit map form and are mapped into relational form for user convenience. (This is an important feature of the RM: whatever form the underlying data takes, the RM presents its user with a relational representation of it. We have coined the term "relationally homomorphic" to describe this.)

- Attribute Level Access

  In addition to relation and tuple management the RM provides routines allowing attribute level access (using the dictionary facilities provided within the system relations). Because the user of the RM does not deal in byte offsets when accessing data, the data can be restructured without affecting existing programs.

- Start, Commit and Cancel Transaction

  Up to ten levels of nested transactions are permitted, and an arbitrary number of RM functions may be included within an individual transaction. In addition, each RM function is transactionally "atomic" by default.

- Distributed Database Capability

  The RM is implemented using server processes which may be allocated on a per-database, per-disc or per-processor basis. This architecture facilitates inter-RM server communications if necessary.

All of these facilities are available via the RM interface and are thus available to all users of sufficient privilege.

4. AFQL - AMPLIFIED FUNCTIONAL QUERY LANGUAGE

The RM thus provides comprehensive facilities for the interrogation and update (hereafter called "query") of individual relations in a database. Most relational query languages allow significantly more than this, for example filtering data using relational and logical operators (including pattern and sound matching), performing arithmetic on attribute values, handling statistical functions such as maxima,minima and averages, returning single-valued and set-valued results from subqueries, grouping results by attribute values, sorting results, filtering out duplicate results, and, most importantly of all, catering for the relational algebra operators **SELECT**, **PROJECT** and **JOIN**. All of this is available within AFQL.

AFQL is a considerable commercial development of the language FQL,[1]. Although the concepts of the two languages are identical, AFQL allows several features not present in FQL, of which the two most important are an update capability and the simultaneous execution of several sequence-reducing functions (such as maxima and averages).

AFQL is best explained using an example. Figure 3 is a procedural schematic of part of a personnel database, which comprises three relations. The STAFF relation contains basic information for each employee, the MANAGERS relation describes the company hierarchy and the DEPT relation describes the company organisation. Domain congruencies (i.e. those domains which may be used during a JOIN operation) are also indicated.

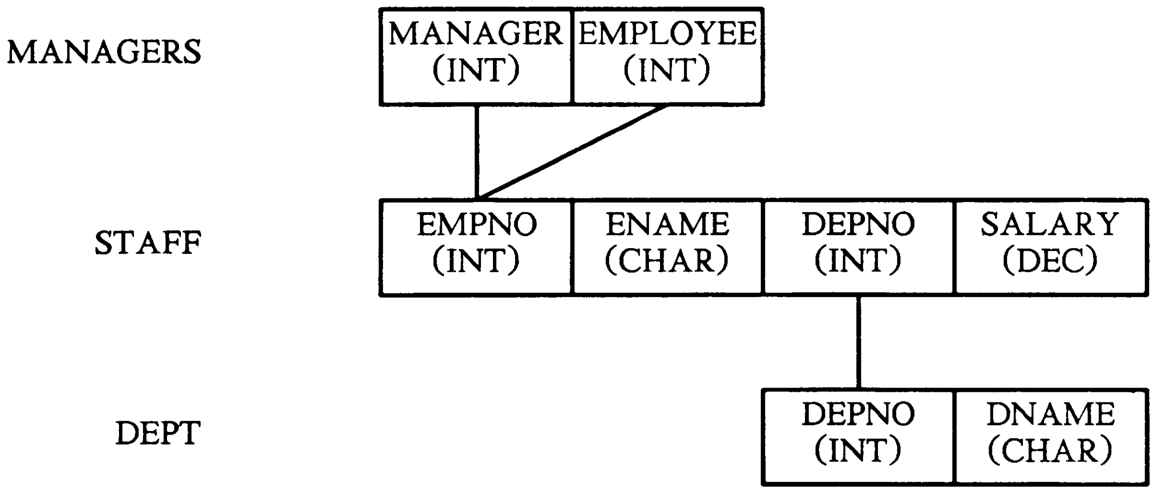


Figure 3. Procedural Schematic of Personnel Database

In AFQL, a database is viewed as a collection of functions which map between relations, tuples, attributes and values, and a query is performed simply by composing these functions (figure 4). For example, in the personnel database **gSALARY** ("get SALARY") is a function, which, when passed a tuple of the STAFF relation (its "domain"), will produce a decimal value (its "range"), while the function **p[SALARY,ENAME]** ("put values into SALARY and ENAME") will place a decimal value and a character string into the SALARY and ENAME attributes of a STAFF tuple.

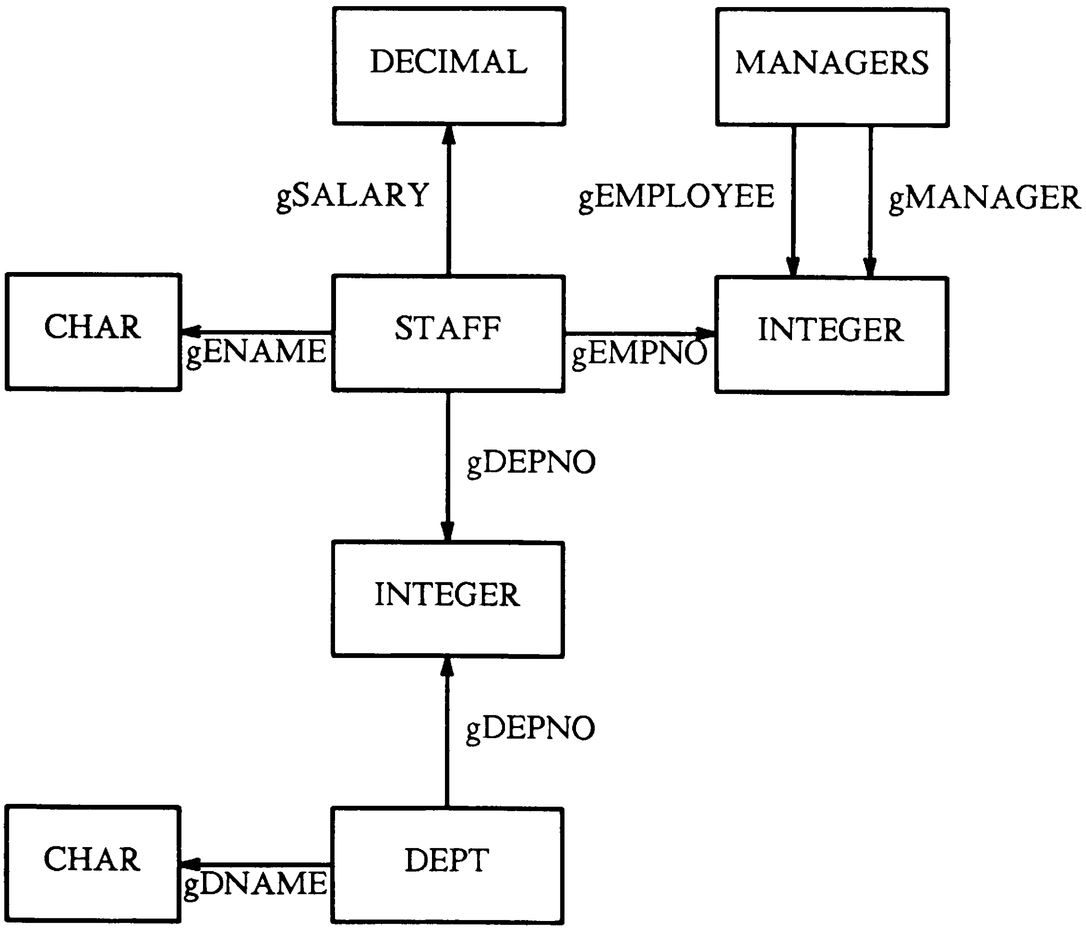


Figure 4. Functional Schematic of Personnel Database

AFQL is best thought of as a database programming language, in which the user concentrates entirely on the domains and ranges of the functions used in the query, and not on variables or assignment statements at all. It may be used in two forms, either as an end user language or as an implementation vehicle for a higher level language. Although its syntax is more suited to computer scientists than the typical end user, it is an elegant language which exceeds the capabilities of most relational query languages available, as the fifth example presented below demonstrates.

The examples are presented with a minimum of explanation and without formal mapping definitions: the appendix of this paper should be consulted for formal details of the AFQL functions used.

(1) English: Who earns more than $20000?

```
AFQL: a[STAFF,SALARY,20000,>]&*gENAME
```

This can be read as "all (a) STAFF with SALARY > 20000, and (&), for each one (*), get (g) ENAME".

In general `a[X,...]` produces a sequence of all tuples in X matching conjunctive simple predicates, while `f[X,...]` produces the first such matching tuple.

(2) English: What are the employee and corresponding department names?

```
AFQL: a[STAFF]&*[gENAME,^DEPT&gDNAME]

where ^DEPT = f[DEPT,DEPNO,gDEPNO,=]
```

^DEPT is a function mapping a STAFF tuple to the corresponding DEPT tuple. The overall query produces a sequence of two element AFQL-tuples, each element being a character string. (The term "tuple" is used in two ways in this paper, firstly as a row of a relation, and secondly as a combination of AFQL functions enclosed in square brackets []).

(3) English: What are the maximum and minimum salaries per department?

```
AFQL: a[STAFF]&*[gDEPNO,gSALARY,gSALARY]&GROUP

where GROUP = grp[max,min]
```

^GROUP is a function which groups a sequence of AFQL-tuples on their first elements, applying "max" and "min" to the second and third elements of these grouped sequences. It produces a three element AFQL-tuple containing the department number, maximum salary and minimum salary respectively.

(4) English: Delete all staff who earn more than their immediate managers.

```
AFQL: a[STAFF]&!([gSALARY,MANAGER_SAL]&gt)&*del

where MANAGER_SAL = f[STAFF,EMPNO,^MANAGER,=]
        &gSALARY
and ^MANAGER = f[MANAGERS,EMPLOYEE,gEMPNO,=]
        &gMANAGER
```

^MANAGER is a function mapping a staff member to his/her immediate manager. MANAGER_SAL finds that manager's salary. In the overall query, STAFF tuples are filtered (!) according to their salary being greater than (gt) their manager's salary (MANAGER_SAL), and these STAFF tuples are then deleted (del).

(5) English: Give all of Smith's staff a 10% pay rise.

```
AFQL: SMITH&JUNIORS&*UPDATE_PAY

where SMITH = f[STAFF,ENAME,Smith,=]
and ^EMPNO = f[STAFF,EMPNO,id,=]
and ^MGR = a[MANAGERS,MANAGER,gEMPNO,=]

&*(gEMPLOYEE&^EMPNO)
and JUNIORS = ^MGR&*([id,JUNIORS]&cons)&red
and UPDATE_PAY = [id,[gSALARY,1.1]&mu]&p[SALARY]
                 &rw
```

SMITH is a function that finds the STAFF tuple for Smith. ^EMPNO is a function that maps an integer X to the STAFF tuple which has employee number X.

^MGR is a function that maps a STAFF tuple to a sequence of all STAFF tuples of that staff member's immediate employees. It achieves this by finding all the employees of the manager, and, for each one, finding the corresponding STAFF tuple.

JUNIORS is a recursive function producing a sequence of all STAFF tuples of all employees below someone in the hierarchy. It achieves this via the "cons" (construct sequence) and "red" (reduce sequence of sequences) list processing functions.

UPDATE_PAY takes a STAFF tuple and multiplies (mu) the salary by 1.1 placing the result back in the SALARY attribute (p[SALARY]) before rewriting (rw) the tuple.

Note that it is impossible to do recursive queries like this through a relational query language such as SQL.

5. IMPLEMENTATION FEATURES OF AFQL

Given the power of AFQL we have just illustrated, the implementation is considerably more concise and compact than might be thought. Some of the major implementation features are described below, many of which are feasible because of the tools and facilities of UNIX and C.

- Yacc/Lex/Sed/Awk

  The syntax of AFQL is defined through BNF. Yacc and Lex are thus used to generate the syntax analysis part of the AFQL translator, and to build the internal evaluation expression corresponding to the external query. Sed is used to alter the yacc/lex externals to different names, so that yacc/lex can be used for the syntax analysis of other languages if desired. By the use of an Awk script it is easy to regenerate the BNF from the yacc source.

- Lazy Evaluation

  A major feature of AFQL is that expressions are only evaluated when they need to be. This is particularly important when there are expensive subqueries, such as those calculating an average. For example, in the query "show employees named Smith who earn more than the average salary for all employees", the average is only calculated if and when the first tuple for a Smith is found.

- Pipes

  Several types of query demand sorting. We have introduced a special sort process which handles several data types, and data is piped to it. AFQL retrieves data from the sort in a manner analogous to obtaining tuples from relations.

- Space Management

  AFQL expressions are held in a list structure in which the nodes have been made of equal size in order to ease space allocation and garbage collection problems. Space for a set number of nodes is obtained from the system via the use of malloc, and maintained internally by AFQL. Further calls may be made to malloc should the evaluation of the query demand it, although this is very unusual.

- Garbage Collection

The garbage collection mechanism uses a reference count in each node and works synchronously with the query. There are thus no long periods when processing stops awaiting the job of the garbage collector. This method is so efficient that, in general, a query needs at most twice as many nodes for its evaluation as it starts with.

- Pointer Manipulation

  C is ideal for the implementation of a list processing language such as AFQL. Considerable use of pointers is made within the AFQL implementation: for example, they are used within the list structure to point not only to other list nodes, but also to the entry points of the AFQL functions themselves.

- Constant Sub-Expressions

  AFQL marks expressions which are known to be constant-valued, so as to avoid evaluating them repetitively. For example, in a query which calculates an average (such as has been described under "Lazy Evaluation"), the average is saved by overwriting the part of the AFQL evaluation structure that calculates it.

- Linked Lists

  Ordinarily, AFQL treats results as sequences in which only the current result is of interest. When handling set-valued results from constant subqueries, however, it links the results together to avoid repeated expensive evaluation. For example, consider the query "show all employees named Smith who earn more than the average salary of at least one department". A typical system will work out the average salary of each department, save the results in a temporary relation, and then compare each Smith's salary with all these tuples. AFQL avoids this by only starting to calculate the averages when the first Smith tuple is found (lazy evaluation), and only calculating as many departmental averages as are necessary to prove whether Smith earns more than some average. These averages are stored together in a linked list as a constant sub-expression. In this manner, should every Smith earn more than the average salary of the first department, the averages of the remaining ones are never calculated.

As a final comment on the applicability of UNIX and C to the AFQL implementation, the code below shows the simplicity of the "sequence construction" function **cons**, used in example (5) above.

```
struct afqls *cons(argp);           /* maps [X,*X] -> *X      */

struct afqls *argp;                 /* pointer to susp. u     */
{
    register struct afqls **tbase; /* general tuple pointer  */
    register struct afqls *node    ;  /* general node pointer    */
    register struct afqls *tup     ;  /* tuple pointer           */

    tup = eval(argp);               /* eval u to give [X1,X2] */

    tbase = tup->nod1.argt;         /* tuple base pointer     */

    node = crstr(*tbase,
            *(tbase + 1));      /* create sequence *X     */

    return(node);
}
```

6. *APPLICATIONS OF AFQL AND RM*

An AFQL interface to an RDBMS, whilst being highly flexible, is not particularly user-friendly. The whole purpose behind implementing such an RDBMS is to provide a tool to be used for the implementation of more useful systems.

Currently we have implemented a Structured Query Language (SQL) interface based upon IBM SQL/DS. The SQL language comprises two basic command types: finite commands and infinite commands. Both types are syntax checked using parsers implemented with yacc and lex, and conventional tree structures corresponding to the query are built.

The finite commands lend themselves to implementation using intermediate AFQL very easily. The command type is used to index into a library of AFQL translations, and the AFQL command corresponding to the current SQL command is retrieved and executed. To achieve this, special AFQL functions performing semantic validation were written. This approach provides an extremely flexible mechanism for the SQL finite commands.

The infinite commands are somewhat more complex, demanding semantic validation such as assigning unqualified column names to their owning tables and incorporating view definitions. This task is handled by special code, after which the "AFQL Builder" routine is called. This routine translates the tree structures into the equivalent AFQL form, following which the AFQL expression is evaluated.

There is no reason at all why the same approach cannot be taken to implement other relational query languages such as QUEL, which is used in INGRES.

Relational query languages are, however, just one of the options available to a systems builder (figure 5). We have plans to use the AFQL interface in several further ways, of which two are outlined below:

— Natural Language

Languages such as SQL are extremely powerful, but are encumbered by having a formal syntax that the user must learn. We are likely to see natural language interfaces increase in importance over the next few years, so that everyone can really take advantage of the relational approach.

One possibility under consideration is to produce a natural language prototype using the AFQL interface. The prototype will access a data dictionary which will include a data lexicon and full semantic information. The user will have the ability to maintain this dictionary, in particular to add synonym type information. The natural language parser will possibly be written in Prolog, building tree structures corresponding to the natural language query and a translator will then transform this into the equivalent AFQL expression.

— Artificial Intelligence

A user has the following problem:

A product, comprising a very large number of complex electronic components, requires maintenance. Whilst experienced electronics engineers are able to analyse and repair most components (whether or not the engineer is familiar with the component), inexperienced engineers are not.

One solution is to set up an RDBMS comprising data describing product descriptions, engineering tools and uses, principles of operation and component lists, using SQL. An artificial intelligence system, implemented upon an AFQL interface to the product data, is used to guide the inexperienced engineer through the maze of component descriptions and test procedures until the product/component at fault is located and repaired. The AI system may choose to modify its test procedures in the light of observed responses.

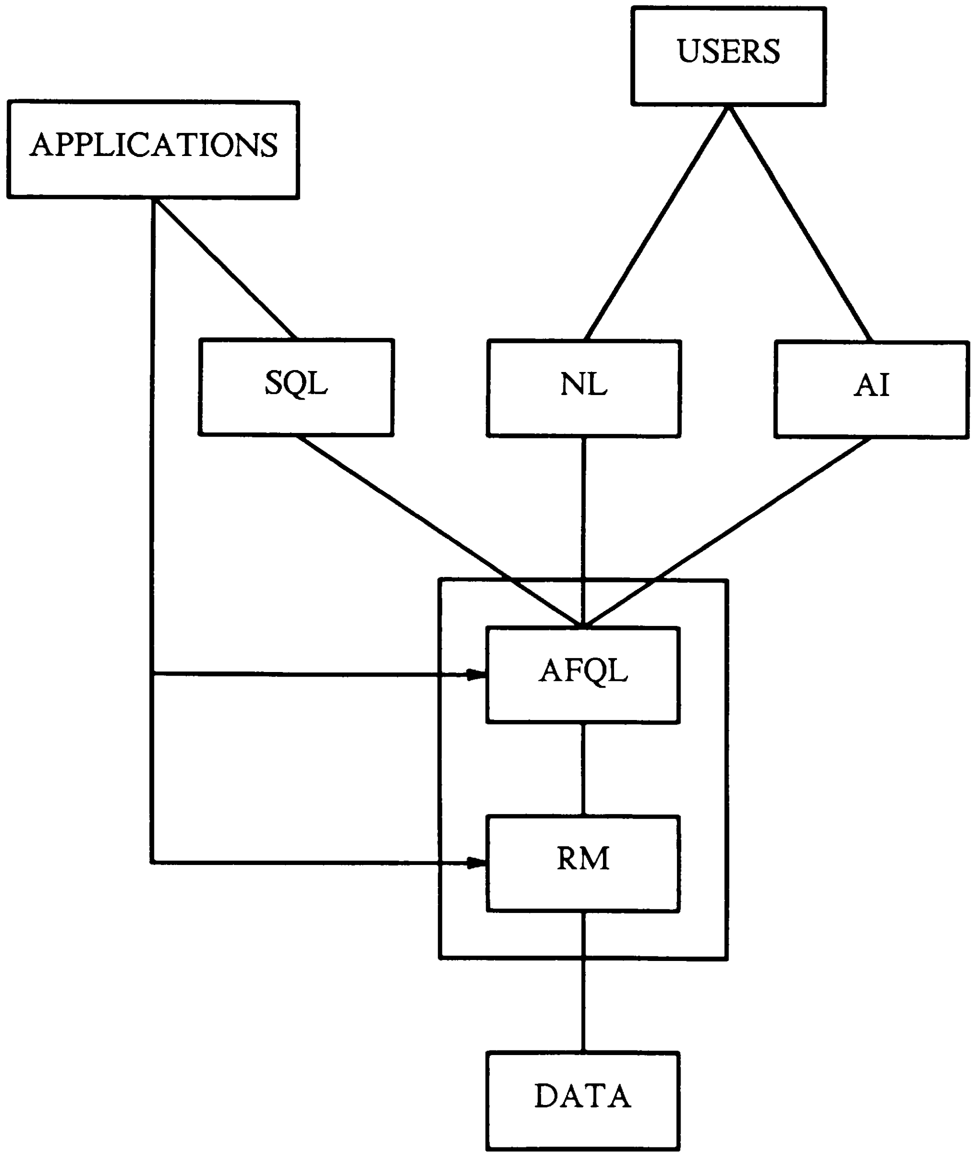


Figure 5. User Applications

7. *SUMMARY*

The traditional hierarchic/networked DBMS is being superseded by RDBMS, particularly in the UNIX world.

Currently SQL is the de facto standard (and the proposed ANSI standard) query language interface to RDBMS.

Man-machine Interfaces are becoming increasingly more sophisticated, and we have illustrated some of the current developments in this area.

Future RDBMSs will have the capability for concurrently supporting several diverse interfaces to a database.

The architecture described provides the tools necessary for the implementation of these diverse interfaces.

8. *ACKNOWLEDGEMENTS*

The significant contribution from Mat Ware, together with the advice and consultation provided by Martyn Legge and Ross MacMillan is gratefully acknowledged.

9. *Appendix: AFQL Definition*

9.1 *Domains and ranges used in the examples*

Entity	Explanation
ROW(RRR)	Row of a relation RRR
RVA	Value of an attribute in a relation with possible value types: SMALLINT, INTEGER, DECIMAL, FLOAT, CHAR, VARCHAR and NULL. Often shown as RVA(TTT) where TTT is the data type.
LIT	Literal with the same value types as RVA. Often shown as LIT(TTT) where TTT is the data type. Where RVA and LIT are alternatives in a mapping, they are shown as RVL(TTT)
TAB(RRR)	Relation name RRR
NAM(RRR)	Column name belonging to relation RRR
CP	Currency pointer (a number corresponding to a dictionary entry defining a particular column of a relation
BOOL	Boolean with three possible values: TRUE, FALSE, MAYBE. This is often shown in AFQL mappings as BOOL(TRUE) etc

9.2 AFQL Operators and Functions used in the examples

FUNC /OP	Name(symbol)	Mapping	Notes
FUNC	AMENDMENT (rw)	rw: ROW(RRR) → ROW(RRR)	Rewrites a tuple to relation RRR
OP	COLUMN ACQUISITION (g)	If V: → CP(?DICT) or V: → NAM(RRR) then gV: ROW(RRR) → RVA	Gets an attribute from RRR. The attribute is identified by a CP for the dictionary row that defines it, or by the column name itself.
OP	COLUMN PLACEMENT (p)	If R: X → ROW(RRR) and Vj: X → RVL and Cj: → CP(?DICT) or Cj: → NAM(RRR) then p[C1,...,Cn]: [R,V1,...,Vn] → ROW(RRR)	Places value Vj in attribute Cj of relation RRR.
OP	COMBINATION ([])	If Vi: X → Yi then [V1,V2,...,Vn]: X → [Y1,Y2,...,Yn]	Each Vi must have the same domain X or may be constant-valued
FUNC	COMPOSITION (&)	If V: X → Y and W: Y → Z then V&W: X → Z	Equivalent to "o" in Buneman's paper
FUNC	CONSTRUCTION (cons)	cons: [X,*X] -> *X	Constructs a sequence
FUNC	DELETION (del)	del: ROW(RRR) → ROW(RRR)	Deletes a row from relation RRR
OP	EXTENSION (*)	If V: X → Y then *V: *X → *Y	Extends function V to operate on a sequence of Xs
OP	FIRST TUPLE ACQUISITION (f)	If T: X → TAB(RRR) and Cj: X → NAM(RRR) and Vj: X → RVL and Rj: X → LIT(CHAR) then f[T,C1,V1,R1,..., Cn,Vn,Rn]: X → ROW(RRR)	Finds the first tuple of relation RRR that matches the predicates defined by the triplets {Cj,Vj,Rj}: see TUPLE ACQUISITION.

FUNC /OP	Name(symbol)	Mapping	Notes
OP	GROUPING (grp)	If F: [X,X] → X then grp[F1,F2,...,Fn]: *[Y,X1,X2,...,Xn] → *[Y,X1,X2,...,Xn] where Y is RVL or ROW	Enables a sequence to be grouped on a value or over the entire sequence. Each Fj is a function to be applied across the group to the corresponding Xj, e.g. "max" to get the maximum of the values
FUNC	IDENTITY (id)	id: X → X	The identity mapping.
FUNC	MAXIMUM (max)	max: [RVL,RVL] -> RVL	Maximum of two values
FUNC	MINIMUM (min)	min: [RVL,RVL] → RVL	Minimum of two values
FUNC	MULTIPLICATION (mu)	mu: [RVL,RVL] → RVL	Multiplication of two values
FUNC	REDUCTION (red)	red: **X → *X	Sequence reduction (equivalent to Buneman's /conc)
OP	RESTRICTION (¦)	If V: X → BOOL then ¦V: *X → *X	Filters the sequence X according to the predicate V
OP	ROW ACQUISITION (a)	If T: X → TAB(RRR) and Cj: X → NAM(RRR) and Vj: X → RVL and Rj: X → LIT(CHAR) then a[T,C1,V1,R1,..., Cn,Vn,Rn]: X → *ROW(T)	Loosely equivalent to Buneman's ! operator with {Cj,Vj,Rj} being simple {column,value, relational operator} conjunctive predicate triplets
FUNC	> COMPARISON (gt)	gt: [RVL,RVL] → BOOL	Three-valued logic

AN EXTENSIVE LINE OF PROGRAMMING LANGUAGES FOR NATIVE AND CROSS DEVELOPMENT WORK

SVS™ FORTRAN 77 / Pascal/BASIC-PLUS/C

SVS family of native mode compilers for MC68000™, 68010™, 68020™, with optional support for 68881 floating point processor. Full ANSI standard, symbolic debugger and optimized code generator with high speed optimization. Support for IEEE floating point, both single and double precision. SVS languages give excellent performance.

u4th

FORTH programming language for UNIX. Largely compatible with the FORTH-83 standard. u4th is interactive and allows full UNIX system call interface, as well as UNIX command pass-through. Permits C primitives and FORTH words to be loaded into a new kernel image. Used frequently in Artificial Intelligence work.

SSI TOOLKIT

SSI Toolkit is a set of Intel-style cross development tools for UNIX and VMS™. Package includes macro cross assembler compatible with Intel ASM-86/87/88/186/188; linker, locator and librarian.

LATTICE® C CROSS COMPILER

Use your VAX™ (UNIX or VMS) or other UNIX machine to create standard MS-DOS™ object code for 8086™ and 186™. The Lattice package includes compiler, linker, librarian, disassembler and 8087™ floating point support. Optional SSI Intel-style cross development tools can be used in conjunction with Lattice for native mode 8086 applications.

AMSTERDAM COMPILER KIT

Now includes BASIC, additional back-ends and simpler licensing!

A package of C, Pascal and BASIC (native and cross) compilers for UNIX machines. Hosted on VAX, PDP-11™, MC68000™, Z8000™ and 8086. Targets for VAX, PDP-11, MC68000™, 6500™ / 6502™, Z8000, 8086, NSC16032™ and 8080 / Z80™. Cross assemblers provided for MC6800™ / 6805™ / 6809™, NSC16032 and Signetics 2650™. Package contains complete sources.

For more information on these and other UNIX software products, call or write:

UniTec
a Division of MEDELCO SA
Postfach 76
CH-3097 Liebefeld/BE
Telephone: 41 31 59 22 23
Telex: 911859

UniTec is the exclusive European Distributor for UNIPRESS and SVS Software

UniTec
a Division of MEDELCO SA

Yet another year with EUUG—S

Bjorn Eriksen
ENEA DATA Svenska AB
ber@enea.uucp

This is a report about the activities within the Swedish UNIX system users group and the UNIX community in Sweden since the spring conference in Paris 1985 —— and perhaps a few bits and pieces from ancient history.

1. EUUG-S ORGANISATION

The Swedish user group started in March 1983 and had its first formal annual meeting in October 1983. The name for the user group was to begin with **SUUG**, but we soon found out that the **Swedish Univac Users Group** already had taken that one. We then came up with the idea to let **EUUG—S** be a short form for **Svenska UNIXsystemanv**

The membership categories and number of members as of April 1986 were

125	Institutional	1000 SKR
15	Individual	700 SKR

About 13% of the institutional members are universities. Obviously more people start companies than universities.

2. EUUG—S ACTIVITIES

So far the Swedish group have had only one major meeting a year. These are arranged as conferences combined with an exhibition. At every meeting a speaker from US has been invited. At the last meeting in September 1985 we had **Brian Kernighan** as a guest. Just over 100 members attended the meeting and we had 10 exhibitors.

The next meeting will be 27-28 October and the guest speaker this time is **Richard "GNU" Stallman**, talking about moral and free software.

In several European countries we start to see a lot of commercially oriented exhibitions, like the one in Paris, Comunix in London and so on. The same thing has developed in Sweden with the so-called "**UNIXfair**". Although this is not a part of the EUUG-S activity as such, the organisers want very much to have the blessing of the national user group, and as we had a booth place free of charge and it's hard to refuse that kind of visibility we have chosen to cooperate with these kind of exhibitions. This fair was attended by more than 3500 people and it will be repeated every year.

We have a local newsletter of course, which is produced at irregular intervals, normally 3—5 a year. In order to improve the newsletter one of the board members was appointed official newsletter editor, and is now constantly struggling with **Wood-syndrome,** trying to find good and interesting articles.

3. EUNET IN SWEDEN

Sweden was hooked up on the network in April 1983 and since then a lot of machines have now joined the net, some have also left it. Just for fun, I've kept the very first electronic mail I ever got from another **Eunet** machine:

```
Return-Path: <mcvax!jim>
Date: Thu,  7 Apr 83 14:02:08 MET DST
From: mcvax!jim (Jim McKie)
To: enea!ber
Subject: Hello

You are now hooked to the mcvax. This is just a test.
Reply, we will be calling you again soon!

Ignore any references to a machine called "yoorp", it
is just a test. Mail should go to "mcvax!....".

Regards, Jim McKie.     (mcvax!jim).
```

I wonder what ever became of that machine **yoorp** ... For about six months there was only one eunet site in Sweden, essentially due to lack of dialers. As I didn't want to buy a DN-11 interface I bought a dialer from **Vecom** in Holland. Actually I had two dialers, one was sent between the Swedish PTT and Vecom in a test-modify loop, the other one I used. To begin with there was one minor problem with a "foreign" dialer as we here in Sweden have the dialing sequence 0,1,2,...,9 instead of 1,2,...,9,0 which is more common elsewhere. This started my career as a uucp hacker. Finally I got it all working and the {\m swnet/} started to grow very rapidly. When I last counted we had

74	registered sites
49	sites connected direct to backbone machine
23	sites receiving netnews

Behind these registered sites there are often a lot of other machines. For example, the **Swedish Defence** have more than 150 sites connected through a separate network with one gateway machine to Eunet.

With the 8 autodialers we can now use 300, 1200 and 2400 baud lines and the X.25 pad connects Sweden to the other backbone machines in Europe as well as **seismo** in the US and even **munnari** in Australia. Well, for the time being munnari can call us but we can't call them. **Mcvax** had the same problem and it turned out that if the PTT routes X.25 through Great Britain, the "No European standard, please, we're British" -island, you loose.

Within **swnet** we also maintain a simple form of "name server", in order for people to easily figure out the electronic mail address of others. A program extracts the information from /etc/passwd and after some hand-editing to remove unwanted names, this list is sent to a special recipient at the backbone machine where the list is in turn sent out to all other sites already in the database. Currently we have 927 names in this "name server" database.

Another in a series of productivity notes on software from UniPress and UniTec

EMACS™

EDITOR FOR: UNIX™/ VMS™/MS-DOS™

Subject: Multi-window, full screen editor.

Multi-window, full screen editor provides extraordinary text editing. Several files can be edited simultaneously, giving far greater programming productivity than vi. The built-in MLISP™ programming language provides great extensibility to the editor.

New Features:

- EMACS is now smaller and faster.
- Sun windows with fonts and mouse control are now provided.
- Extensive on-line help for all commands.
- Overstrike mode option to complement insert mode.
- New arithmetic functions and user definable variables.
- New manual set, both tutorial and MLISP guide.
- Better terminal support, including the option of not using unneeded terminal drivers.
- EMACS automatically uses terminal's function and arrow keys from termcap and now handles terminals which use xon/xoff control.
- More emulation-TOPS20 for compatibility with other EMACS versions, EDT and simple Wordstar™ emulation.

Features:

- Multi-window, full screen editor for a wide range of UNIX, VMS and MS-DOS machines.
- «Shell windows» are supported, allowing command execution at anytime during an edit session.
- MLISP programming language offers extensibility for making custom editor commands! Keyboard and named macros, too.
- «Key bindings» give full freedom for defining keys.
- Programming aids for C, Pascal and MLISP: EMACS checks for balanced parenthesis and braces, automatically indents and reformats code as needed. C mode produces template of control flow, in three different C styles.
- Available for the VAX™ (UNIX and VMS), a wide range of 68000 machines, AT&T family, Pyramid™, Gould™, IBM-PC™, Rainbow™ 100+ and many more.

TELIX

TELEX CONTROL UNDER UNIX

ELIMINATED:

- PAPER TAPE
- WRITING and MODIFYNG telex messages on telex terminals with limited capability.
- INTERRUPTIONS while trying to send a message due to incoming or other «important» outgoing message(s)

ADVANTAGES:

- Multi-user Unix systems allow many users to prepare and submit telexes simultaneously. TELIX resolves all scheduling and priorities.
- Incoming telexes stored in computer for easy reference. Any part of incoming telex may be easily included in another transmission.
- TELIX system includes user-friendly «menu interface».
- Extremely easy to work with TELIX.
- Training time for new users about 30 minutes.

SOPHISTICATED FEATURES:

- Telexes may be prepared and transmitted at specified later time, allowing telex line to be left open during business hours.
- Priority system used in the queue = important telexes may be sent ahead of others.
- Consolidate numerous telexes for same destination for one transmission; eliminating repeated calls to same number.
- When a call fails for any reason, telex left in queue to be tried again in about 30 minutes with lowered priority.
- Any number of lines managed simultaneously.

Q-CALC

SPREADSHEET

Subject: Powerful spreadsheet with NEW ADDED FEATURES.

Q-Calc is an extraordinary spreadsheet for UNIX including extensive math and logic facilities, comprehensive command set, optional graphics, many new ease-of-use features, and the ability to run UNIX programs on spreadsheet data.

Features:

- Fast spreadsheed with large model size, allowing sorting and searching.
- Interfaces with UNIX and user programs via pipes, filters and subprocesses. Data can be processed interactively by UNIX.
- Q-Calc profile mechanism allows the user to store default information, as well as support for terminal-specific profiles. Uses termcap.
- Available for Plexus™, VAX™, Sun™, Masscomp™, AT&T 3B & 7300 Series, Pyramid™, Gould™, Cadmus™, Integrated Solutions™, Cyb™, IRIS™, Callan™, and many more.
- Graphics for bar and pie charts. Several device drivers supported.
- New Features of Version 3.2 include more powerful printing, simpler data input, keybinding definitions, new string operator, bind-to-key, and more.

For more information on these and other UNIX software products, call or write:

UniTec
a Division of MEDELCO SA
Postfach 76
CH-3097 Liebefeld/BE
Telephone: 41 31 59 22 23
Telex: 911859

UniTec
a Division of MEDELCO SA

A Study in Digital Image Reduction or How to Make Small Faces

Doug Kingston

Centrum voor Wiskunde en Informatica
Amsterdam, The Netherlands

dpk@mcvax.uucp
dpk@brl.arpa

1. INTRODUCTION

Those of you at the Florence conference this spring are likely aware that there were several people cajoling attendees to get their picture taken for something called "the faceserver". Those who managed to get to the I2UNIX booth on the second floor found a group of energetic Germans from UNIDO with a large television camera and a table full of bits and pieces of Atari 520st and related peripherals. The purpose of this equipment was to collect digitized faces for later conversion into small bitmap icons.

You might ask why would someone want to collect faces. There are many uses for the face icons, but they all have in common the desire to have graphical representations for people that can be used in place of the traditional written names or addresses. Associating an identity with a picture can often be easier than associating the identity with a written name especially when the name is abbreviated (e.g. a login name). The faces can also be used in a "white pages" service to allow us match a face with name or to help remind us which person goes with what written name.

The first work on face icons was done by several members of Bell Labs, Murray Hill, in particular Rob Pike and Tom Duff. They found that a useful representation of a face could be gotten in an icon no larger than 48 by 48 bits. They spent a good portion of the Utah Usenix conference collecting faces with a 35mm camera. Their work in this area stimulated a number of other people to look into doing the same thing. At the Dallas /usr/group exhibition, Masscomp had a booth that included a Masscomp system set up to collect faces and store them with the person's name and net address.

The most technically interesting aspect of these face icons is that they represent the output of a very large data compression operation. The original images are typically 256 by 256 by 4 bit grey level images. In some cases the original images have even more resolution. When converted into icons, the information in that image (32K bytes) is compressed into 288 bytes (48 by 48 by 1 bit)! Now needless to say there is significant information loss here, but if you are clever you can maintain a very recognisable image despite the information loss. *How* you compress the data is the crux of the problem.

2. *FACE ICON FORMAT*

The work at Bell Labs has developed a standard of sorts for face icons. The format for interchange is an ascii representation. The icon description consists of 48 lines each specifying 48 bits. Each line consists of three comma separated hexadecimal numbers with no white space. Each number specifies a 16 bit word in big endian order. For a given word, such as 0x1267, the output bit horizontal bit pattern would be: 0 0 0 1, 0 0 1 0, 1 1 0 0, 0 1 1 1. Put verbally, write out the the binary representation of each nibble from left to right, most significant bit first.

The following is the complete description of the face of Peter J. Weinberger. His face is a somewhat of a standard of its own. I have include the entire description as a reference. If you get this face right, you can view any face. (This description would normally be single column. It has been listed in multiple columns to save space. The first line contains scanlines 1-3, read from left to right, the second line contains scanlines 4-6, etc.)

```
0x0000   0x1FA0   0x0000   0x0000   0x7FFC   0x0000   0x0000   0xFEFF
0x0001   0xBFBF   0xC000   0x0001   0xEFFA   0xE000   0x0002   0x00FF
0x0002   0x007F   0xFF80   0x0000   0x001F   0x7E00   0x000C   0x001F
0x0000   0x0015   0xFFC0   0x0030   0x001F   0xFFF0   0x0070   0x000F
0x00C0   0x0007   0xFFF8   0x00E0   0x0007   0xFFE8   0x01E0   0x0003
0x03C0   0x0001   0xFFF8   0x03E0   0x0001   0xFFF8   0x07C2   0x8000
0x0FFA   0xF83F   0xBFF8   0x0FE0   0x7C67   0xFFF8   0x1FF1   0xEFF9
0x1FF7   0xFFFF   0xFFF8   0x1FF5   0xA4FF   0xFFF8   0x1FD0   0x247F
0x0FE0   0x8045   0xFFF0   0x1FA0   0x0870   0x3FF0   0x0320   0x0060
0x0B80   0x1030   0x1F80   0x0181   0x601F   0xF780   0x0085   0x405A
0x0000   0x0BFB   0xFF00   0x0180   0x1FE8   0xEE00   0x0380   0x07F2
0x0182   0x03D5   0xEFC0   0x0080   0x0075   0x7F80   0x01C1   0xD5DD
0x00C1   0x57FF   0xFF80   0x0040   0x0039   0x7F00   0x0060   0x006B
0x0004   0x1FEB   0x6000   0x0000   0x07FF   0xF800   0x0002   0x0015
0x0000   0x0057   0xC000   0x0002   0x003F   0x4000   0x0000   0x804B
0x0002   0xBBFE   0x8000   0x0000   0x8FFB   0xC000   0x0001   0x7ABF
```

3. *OUR EUROPEAN FACESERVER PROJECT*

Since Dallas, little has happened until slightly before the Florence conference when I rekindled the idea after briefly mentioning it to Daniel Karrenberg. Much to my surprise, UNIDO immediately went out and got an inexpensive video digitizer for one of their many Atari 520st's, and we then began a crash program to get enough face server collection software running to take it to Florence. We had two weeks, but we made it. A large portion of the credit goes to Hans-Martin Mosner whose work on the collection software for the Atari made the idea a reality in time for the conference.

The actual hardware was:

— an Atari 520 ST+ (1MByte)

— 2 double sided FD drives (3 1/2 inch)

— b/w television camera (really old stuff)

— video digitizer

Our software made heavy use of the GEM user interface on the Atari. Data was read from the digitizer and displayed in 4 windows:

256*256*1	(thresholded raw data),
48*48*1	(resulting icon),
48*48*1	(magnified 4 times for bitmap editing),
	and a histogram of grey value distribution.

Since we had too little time to develop compression algorithms, the icons looked quite ugly. Most people couldn't imagine that it was their picture we took, but we actually stored more data than was displayed. If you have questions about the technical details, feel free to ask hmm@unido.uucp.

At Florence we managed to collect around 150 faces in three days. However, it was a learning experience. The equipment had necessarily been put together in a hurry and we had a couple of hardware failures (broken cables, flakey floppy drives). We also were not equipped to properly light those being photographed, and the lighting in the room was quite variable. I don't think we realized at the time how critical the lighting was. This problem will most certainly receive more attention in the future. As a result not all of the faces are usable, but we have done our best to have our digitizing algorithm compensate to the lighting automatically (see the algorithm description below).

We had originally intended to only store the compressed icons but after a short while we were convinced that our current algorithms were not good enough. We decided to store the full 32K byte images on multiple disks and process them after the conference when we had time to refine the data compression algorithms. This has been a low priority task for all of us, so things have moved slowly until recently when more time could be devoted. We now have a quite good compression algorithm, and we will hopefully be able to seed the European faceserver with the faces collected at Florence. Those who had electronic mail addresses on EUNET should receive an ascii copy of their face by the end of August 1986. The entire collection should be made available in the not to distant future, although the exact method is not yet known.

4. THE DATA COMPRESSION ALGORITHM

Some explanation is necessary of the problems in getting good face icon representations. Initially, it may be necessary to correct for the aspect ratio of the digitizer vs. the aspect ratio of the camera. In our case the camera's 4 by 5 aspect ratio was scanned into a 256 by 256 raster. The picture can be divided into several distinct portions each with different qualities. The background of the picture is white. It should stay white since it serves to outline the face. Shadows must be avoided on the background. Very bright features and very dark features should stay that way. In general it is fairly straightforward to assign the very bright and very dark areas to 1 and 0 respectively. The real problem in the compression process is to maintain "greyness" in the image. Since you have only 288 output bits (48x48), you cannot use dithering in the classical sense. You simply cannot afford to take several bits to represent one value. Every bit in the icon must be chosen so that it helps convey the sense of the original image. Since you have only 0 and 1 as values for each pixel, you can only introduce "greyness" by considering how a given pixel interacts with adjacent pixels. The human eye will not normally be close enough to pick out individual pixels of the icons so you can use the natural tendency of the eye to average a given pixel with its neighbors. This can be exploited to produce very good icon representations of 48 by 48 grey level images in 48 by 48 icons.

We realized in Florence that the basic problem was reproducing the grey in images. I spent a considerable amount of effort on trying to reproduce grey in the icons with generally poor results. The first attempts tried to use weighted calls to a random number generator to decide if the bit should be on or off, but the information loss from the random factor was not acceptable and it did not take into account the neighbor averaging effect. The next algorithm was concocted by myself and Hans-Martin while waiting for an EUUG Executive meeting to adjourn in Florence. The idea was to set all the known

1's and 0's, and then to set each remaining pixel based on the grey value desired, and the current surrounding pixels. The evaluation was left to right and top to bottom in that order and unfortunately produced a very regular striping of the image. The next round of attempts focused on using prime numbers to randomize the evaluation of the image without in fact being random. This also produced patterns although they were less describable. The biggest strides on improving the algorithm came on the weekend of 21 June when I was making a visit to Dortmund. Sunday afternoon Daniel Karrenberg and I spent about 5 hours improving the algorithm. The key was the choice of averaging overlay. We had been using a 3 by 3 grid with the pixel of interest at the center. The flaw in this type of approach is that the pixels already assigned unduly bias the next assignment, and do so in a geometrically regular manner. The current version of the algorithm is not susceptible to this regular distortion. We now use cross shaped averaging overlay that examines the four pixels immediately adjacent to the image (above, below, right, and left). The image is scanned 4 times examining a quarter of the pixels each time. The order is (even,even), (odd,odd), (even,odd), and (odd,even).

What follows is the current algorithm with a number of comments.

```
int  value(x,y); /* return grey value from original image */
int  map[];           /* index by grey value to get new grey value */
long face[48][48];    /* summed grey values */
int  buckets[]; /* histogram counts */
```

These are the basic data structures. Since the target icon size is 48 by 48, we can conveniently compress a 192 by 192 image into the required 48 by 48. This conveniently drops the left and right side of the image which are unnecessary border. The following chunk of code also maps the 256 vertical pixels into 192 as a side effect.

```
/* Fix aspect ratio and sum the grey values into 48x48 array */
for (y = 0; y < 192; y++) {
      register int val;

      for (x = 32; x < 224; x++) {
            val = map[value(x,(y*256)/192)];
            face[y/4][(x-32)/4] += val;
      }
}
```

So now we have a roughly 48 by 48 by 8 bit image. The next basic operation is to determine the mean grey value and "shift" it towards the center of the range. This is basically a contrast enhancement step.

```
    /* Determine the mean */
    for (y = 0; y < 48; y++) {
         for (x = 0; x < 48; x++)
              mean += face[y][x];
    }
    mean /= (48*48);

    /* Actually do the shifting */
    mean = tmean - mean; /* Convert to shift amount */
    for (y = 0; y < 48; y++) {
         for (x = 0; x < 48; x++) {
              n = face[y][x]/(MAXSUM/NBUCKETS);
              if (n >= shiftlo && n <= shifthi) {
                   face[y][x] +=
(mean * (2*tmean - abs(face[y][x] - tmean)))/(2*tmean);
                   if (face[y][x] < 0)
                        face[y][x] = 0;
                   else if (face[y][x] > MAXSUM)
                        face[y][x] = MAXSUM;
              }
         }
    }
```

We now determine where the low and high water marks should be. The trick here is to do it based on a percentage of the total number of pixels (a percentile).

```
for (y = 0; y < 48; y++) {
     for (x = 0; x < 48; x++) {
          n = (face[y][x]*NBUCKETS)/MAXSUM;
          buckets[(n >= NBUCKETS ? NBUCKETS-1 : n)]++;
     }
}
n = 0;
x = 0;
while (n < ((48*48)*lowater)/100)
     n += buckets[x++];
lowater = ((x-1)*MAXSUM)/NBUCKETS;
while (n < ((48*48)*hiwater)/100)
     n += buckets[x++];
hiwater = ((x-1)*MAXSUM)/NBUCKETS;
```

Having done this, we are now ready to start assigning bits to the icon. The first and easiest step is to assign the bits that are so dark or so light there is no question as to their appropriate value.

```
/* Wire down the known bits */
for (y = 0; y < 48; y++)
     for (x = 0; x < 48; x++) {
          if (face[y][x] < lowater) {
               face[y][x] = 0;
               /* clrbit(x,y); */
          } else if (face[y][x] > hiwater) {
               face[y][x] = MAXSUM;
               setbit(x,y);
          }
     }
```

Now for the hard ones. We walk around the image in a regular but carefully chosen manner so that the effect of previously assigned pixels does not generate feedback patterns. The first two passes run into only preassigned pixels and no others. The second two passes will always have all neighbor pixels assigned.

```
/* pass1 get every other pixel (even,even) */
for (y = 0; y < 48; y+=2) {
     for (x = 0; x < 48; x+=2) {
          if (face[y][x] == 0 || face[y][x] == MAXSUM)
               continue;
          choice(x,y);
     }
}

/* pass2 get every other pixel (odd, odd) */
/* pass3 get every other pixel (even, odd) */
/* pass4 get every other pixel (odd, even) */
/* ... same as above, just change subscripts. */
```

The choice function decides which value to set the pixel to based on which one will more closely approximate the desired grey value.

```
choice(x,y)
register int x, y;
{
	long sum();
	register long ifzero, s, a;

	ifzero = sum(x,y) - face[y][x];/* what zero gives you */
	s = (2*ifzero + MAXSUM)/2;     /* dividing line */
	a = 5*face[y][x];         /* what you want */

	if (a > s) {
		setbit(x,y);
		face[y][x] = MAXSUM;
	} else {
		/* clrbit(x,y); */
		face[y][x] = 0;
	}
}

long
sum(x,y)
int x, y;
{
	static intxs[5] = {0, -1, 0, 1, 0};
	static intys[5] = {-1, 0, 0, 0, 1};
	register int xx, yy, i;
	register long s = 0;

	/* sum over the 5 pixel in a cross about this pixel */
	for (i = 0; i < 5; i++) {
		xx = x+xs[i];
		yy = y+ys[i];
		if (xx < 0 || xx > 47 || yy < 0 || yy > 47)
			s += face[y][x];
		else
			s += face[yy][xx];
	}
	return(s);
}
```

There are several important tunable variables in the algorithm. The first is the input map. I am current using the following map: 0 1 2 2 3 5 7 9 10 11 12 13 14 15 15 15. The lowater and hiwater marks are points which control when pixels get hardwired to zero and one respectively. Each is expressed as a percentile after the pixels are shifted. I am currently using a lowater mark of 25% of and a highwater mark of 55%. The range of pixels subject to shifting is tailorable, however I am currently shifting all pixels. The tmean is the "target mean". This is the value towards which we try to shift the "mean" value. It is expressed as a percentile based on the original (unshifted) pixel values.

We are happy to make available any of the software we have written to do this work. In particular the complete version of the above program should be usable on any UNIX system since it only uses read, write and stdio. It's just a filter. We also have a simple histogram program for helping to analyze an image's information density. We will mail these to any interested parties. Simply contact us by electronic mail. We have some other tool-like programs for use with both the DMD5620 and the Atari 520st. Tools for making icons on the Whitechapel MG1 from our binary files exist but need to be collected. I will make the DMD software available, and the Atari stuff can be gotten from hmm@unido.uucp.

SPONSORED UKUUG MAILSHOTS

Are you interested in a fast, economical way of reaching around 300 named users of the UNIX System in the UK?

If you are, why not make use of the regular **UKUUG MAILSHOT** which can take your literature or message direct to these key people?

The cost to Mailshot sponsors is reasonable — make the most of this opportunity and contact us NOW for further details.

The Secretariat
UKUUG
Owles Hall
BUNTINGFORD
Herts SG9 9PL
England
Tel: +44 (0) 763 73039
Net: euug@inset.uucp

UNIX and the electronic office - cognitive ergonomic reflections

Gerrit C. van der Veer

Amsterdam

1. INTRODUCTION

This contribution concerns a special kind of computer user. In offices, computers are becoming increasingly common. The modern office, often called an "electronic office", has many tasks allocated to electronic devices, to either individual special purpose devices or to a general purpose system.

The UNIX operating system evolved in academic and laboratory situations. UNIX got its start as an operating system for text processing programs. Although UNIX was the developers' goal, for Bell Laboratories it was an acceptable byproduct of programs to complete copyright applications. That is, UNIX has always been heavily oriented towards text processing. At present, however, several developments point in the direction of other types of use and other groups of users.

- The X/OPEN activities of the collective major European computer manifactors aims at portability and usability of UNIX in a multitude of different applications and environments. Their systems find their way to "open shops" in which non-expert users are doing, among other things, all kinds of administrative jobs.
- In the software industry an increasing number of both small and large companies are developing their products in a UNIX environment. Turnkey systems and custom-made software tools may be constructed on UNIX, including a lot of office applications.

The growth of UNIX based applications for offices and office tasks leads to new kinds of problems for a new group of users.

2. UNIX IN THE OFFICE ENVIRONMENT

There are good reasons for the growing popularity of UNIX for office applications.

- The standard text processing facilities are powerful, although not easy to handle for incidental and novice users.
- Communication between systems, users and terminals is readily available (**mail**, **write**).

- The hierarchically ordered file system is very simple and elegant.
- The multi-user and the multi-tasking concept are combined in a logical way.
- By producing applications with the "standard" UNIX environment in mind, they gain in portability and flexibility, allowing customers to change to new hardware and new organisational structures.
- Nearly "anything" that might be useful in an office situation may be constructed in a UNIX environment, although advanced understanding and good programming strategies are essential.
- UNIX compares favourably with other general purpose operating-systems for mini computers and mainframes, especially for non-professional users.

Users in an office situation are not normally computer professionals. They are experts in their own task domain: management, typing, secretarial jobs, administration. In the light of this situation Norman (1981) wrote an evaluation of the possibilities for non-expert users to work with the UNIX system, with the title "The trouble with UNIX". Although he did not consider any other operating system worth commenting on, he called UNIX "a disaster for the casual user", and the criticism he casts upon the system from the point of cognitive psychology has to be taken seriously. His main points are:

- inconsistency of command names, functions and syntax.
  - The command **cat** may be used to list the content of a file on the terminal, whereas the name **cat** is short for concatenate, which presupposes at least two arguments.
  - The symbols /, **$** and * have several different meanings, / even within a single expression.
  - Commands like **ln** and **mv** are abbreviations of the words "link" and "move", words that are of the same length as the commands **mail**, **date** and **kill**, which are written in full.
  - The first two characters of **chdir** denote a change, but the command for "change password" is **passwd**.

  These inconsistencies are a source of errors and misunderstanding for non-expert users. Commands should be easy to remember, even if they are not used very often. Names should help as a mnemonic aid, they should be systematic in order to be reconstructed by analogy, and have the right semantic connotations, referring to the functions they are used for.

- user unfriendliness.

  When a command is accepted, UNIX remains silent, even if the result has probably not conformed to the user's intention, as in **rm chapter ***. If, however, the system cannot decide about the intention of a command, the feedback is usually a "usage" line, disapproving the user's action. This "quiet" approach, on the other hand, is one of the most appealing aspects for a regular user: messages only will appear if there is a good reason to react to them.

3. *DESIGN CRITERIA FOR AN OFFICE SYSTEM*

Office systems are constructed for a special task domain. The typical user can be expected to be an expert in this domain, but may be not an expert in computing in general (e.g. may not be a professional programmer). Office systems differ from other systems for this user only in regard to the task domains. Apart from the need for application dependent tools, the general design criteria are the same.

3.1 *Ease of use*

In relation to this criterion several aspects are relevant:

i. Commands can only be used if their name and their syntax can be remembered. Both the names and the syntax should be regular and systematic (so they can be reconstructed), and the names have to be chosen in such a way that they refer to the meaning. There are many ways to solve this. One of the simplest is to make an *additional* entry with the "improved" name via the command **ln** (link). This is the preferred way because no extra disk space is required and management is simplified. Change the contents of the old bad name, and the new name is changed too. A second way is **mv** (move) the old name to a new name. This is confusing for expert users. A third way is to **cp** (copy), but this wastes disk space and does not guarantee that both programs are always identical. A fourth way is to write a shell script that, among other things, calls the original program.

ii. If a user needs help this should be available on the level at which the problem exists: Moran's four level command language grammar (1981) makes a clear distinction between task level, semantic level, syntax, and key stroke level. Help is only useful if it describes the facility on the level at which the user needs clarification. The user should be able to specify this level if desired.

iii. The number of symbols to be typed, and especially the amount of bizarre combinations of key strokes makes a difference for the casual user. Unusual combinations of actions in combination with insufficient feedback often lead to errors. On large general purpose mainframe systems a considerable number of sessions is aborted even before the actual start, because of these kinds of errors during the login transaction (Coombs and Alty, 1981).

iv. Offices differ from each other. Occasionally a certain office is characterised by a frequently used special task, for which a "tailor made" command is of great help. The system should allow an easy construction of this kind of special purpose facilities.

v. Users differ from each other too. For example, programmers usually like the "silent" approach. For them the computer is there to worry. Too much feedback would complicate reading program output. For them programs should not say something without being asked. There should, however, be "verbose" options for users who need this. For example, **tar** is normally silent, but many users add **v** for the verbose output.

3.2 Resistance to slips

By slips we refer to errors due to incorrect actions even though the user's original intention is correct, e.g. typing errors such as permutation of characters, the interchange of control-key and shift-key, the use of upper case symbols in the editor **vi** when a lower case command symbol is intended (which is often only discovered after a few key strokes, when the damage is irreparable).

Slips may be avoided or at least made less dangerous. Consistency of commands will help to avoid confusion, commands with different results should not be nearly identical sequences of key strokes, like the **vi** commands **:w** (write) and **:wq** (write and quit, a combination of two commands that could originally only be issued separately). This last command may be replaced in later versions of UNIX **vi** by **ZZ** (a command issued without temporally leaving the visual mode), which will lead to less confusion with **:w**. This command, however, is incomparable to other commands that have semantic relations outside the scope of **vi**, by not being preceded by **:** and by not needing to be terminated with carriage return. Moreover, the mnemonic value of **ZZ** is rather questionable.

For all kind of slips an undo possibility is in fact unavoidable. Thus far it is not implemented in most UNIX facitilies. The shell itself is a command interpreter, and can't redo the actions of commands external to it. The shell should be extended to include this facility.

As far as confusion is created by the occurrence of different modes (**vi** and **ex** as two modes of one editor), this feature has to be either avoided, or presented very explicitly and consistently to the occasional user.

3.3 Mental models of the system

Norman (1983) makes a distinction between the system, the conceptual model of the system, and the mental model the user has in mind. The conceptual model is, in fact, a description of the user interface: the part of the system the user should be aware of in performing his task in interaction with the system. The conceptual model of a system depends on the task domain and on characteristics of the group of users. Only when his mental model is in accord with the conceptual model will the user have correct expectations concerning the outcome of his actions.

The relation between the conceptual model of the system and the mental model in the mind of the user is dependent upon two aspects:

i. The interface is designed after some conceptual model. Only if this conceptual model is constructed with the task and the special group of users in mind is it possible for those users to develop mental models of the user interface in which the other part of the system remains invisible. The user only has to take note of those aspects of the system that are directly relevant to his view of the task domain.

ii. Interaction with the system is accompanied by a special kind of communication, the content of which is the interaction itself. We call this type of communication metacommunication. Different forms of metacommunication will take place, some of them initiated by the user, such as asking for help and looking for documentation. The **man** command is an online facility in UNIX that does not present a description, which a novice can understand without having to ask an expert to translate it for him. Apart from this the **man** command can only be used if one knows the correct name of the command on which one needs help. The UNIX programmers manual (from which the content of **man** is derived) is definitely not a primer (the alphabetic order and the "permuted index" have no relation to any sensible structure), so the beginning user needs other kinds of metacommunication to learn new facilities and discover unknown possibilities.

    A special kind of online metacommunication may be called implicit metacommunication. This consists of the actual names of commands or the appearance of icons. Command names always suggest some meaning, derived from the common semantic associations with the word. In the editor **vi** a group of commands containing the character **d** suggest the meaning "delete" although in fact the action performed is the transport of some information from the edit file to a single buffer. The information remains available for repeated copy-and-insert commands and is only erased by refilling the buffer.

The above mentioned aspects of user friendliness are strongly interrelated, especially when novice users are involved.

4. ADAPTATION OF UNIX FOR THE ELECTRONIC OFFICE

In this section we will present some examples of new developments that may add to the usability of UNIX for casual and non-professional users in office environments.

i. New commands and facilities have been added to UNIX, although this is one of the reasons different variants arise. Some commands indeed result in more user friendly systems, coping with the objections we mentioned in the last section. Other new commands are however only powerful tools for experts. An example of a new feature that certainly is an improvement for some users is **vsh**, a visually oriented shell with a menu structure.

ii. Special applications often ask for dedicated facilities. Tailor made software is always the best method to cope with individual wishes, taking care of the problems of casual users. In situations in which programming experts are accessible, e.g. in large offices and in university environments, these kinds of tools can be constructed. In fact, UNIX contains many possibilities needed for these kinds of extensions. We know of examples such as dedicated spreadsheet-like systems, data retrieval and library programs, constructed according to specifications given by administrative workers. These users have expert knowledge of their task domain and have learned to specify their needs in such detail that a programmer can do the job. These examples, however, are only to be found in environments of relative luxury, where programmers are available and cooperative. This situation might, in fact, result in incompatible systems, if programming is done at system level, destroying the real profit of portability, but it can be done at a higher level. A danger of this method is that it invites programmers to re-invent the wheel.

iii. The novice user needs the opportunity to develop his own mental model of the system. Therefore metacommunication is indispensable. The original UNIX facilities are insufficient for users with a limited knowledge of the system. The system UC (UNIX Consultant) (Wilensky, Arens, Chin, 1984) offers the possibility for users to ask the system questions in natural language about the semantics and syntax of UNIX commands, by consulting an Artificial Intelligence user interface and a database derived from the UNIX documentation. This facility is a useful alternative for the UNIX programmers manual (Kernighan, McIlroy, 1979) that is of no help for some users. It may prevent them from wandering around the office, looking for colleague victims to discuss problems none of them is really capable of solving, giving each other the idea that they at least understand the problem. For professional programmers the UNIX programmers manual can be an efficient source, without unnecessary garbage, packed with exact information.

iv. In situations in which a user needs only a limited set of facilities, such as extended text processing, or only telecommunication possibilities, task-specific user interfaces may prevent the user from getting lost in the multitude of possibilities of the UNIX system. This situation is often a temporary one, in that the user, after having gained sufficient experience with his main task, will start to use the system for additional applications. The interface can only be constructed after the task level and semantic level of the discourse (Moran, 1981) have been analysed, i.e. what is "logical" for the naive computer user. Research by a combination of European centers of human factors and informatics (van der Veer, Tauber, Waern, van Muylwijk, 1985, Hannemyr, Innocent, 1985) aims at designing a user interface for the UNIX mail system. Starting point will be an analysis of the office mail task space. The interface will be constructing in SYNICS (Edmonds, Guest, 1984), a tool for dialogue design, with syntactic and semantic rules to translate user input to system commands. With this kind of interface all official possibilities of UNIX are preserved, as should always be a goal, but the special task domain can be handled with a communication protocol that appears logical to the user and presents optimal metacommunication.

v. In some situations it is not sensible to expect users to communicate with the UNIX system directly via the Bourne shell, even if the special possibilities and the architecture of the shell are well suited for the application domain. In that case a user interface may be defined which completely hides the original system, and is optimally oriented to a task specific way of interaction, e.g. screen oriented form-filling, or the use of menus, with task oriented feedback and hierarchical control structures based on the structure of the task domain.

UNIKIT (CERG, no date) is a powerful tool for the creation of task environments for administrative jobs. The use of this tool does not in itself guarantee a design that is justified in cognitive ergonomic science (as in the case with SYNICS). In the case of UNIKIT research is now going on in which design guidelines will be incorporated in a database (written in PROLOG) in order to supply the designer with criteria to evaluate his prototype in the course of construction.

We expect UNIX will become popular in office situations. Users in these types of environments are not computer experts, so they need special facilities. The examples from this section illustrate ways to enhance the user interface, as long as these methods are applied with a clear view of the human computer interaction in mind (e.g. aiming at ease of use by defining short and handy commands), taking into account the effects of the method on other aspects of the interaction (e.g. the mental model that is introduced or adapted by the semantics that are derived from the mnemonic code), and preserving the unique characteristics for which the UNIX system became famous.

5. *REFERENCES*

CERG (no date)
UNIKIT version 1.1 Editeurs d'ecrans. CERG, Groupe BFM, Paris.

Coombs M.J., Alty J.L. (Eds.)(1981)
Computing skills and the user interface Academic Press, London.

Edmonds E., Guest S. (1984)
User Guide for SYNICS Leicester Polytechnic, Leicester, U.K.

Hannemyr G, Innocent P.R. (1985)
A network user interface Behaviour & Information Technology, 4, p. 309-326.

Kernighan B.W., McIlroy M.D. (1979)
Unix programmer's manual Bell Telephone Laboratories, Murray Hill, N.J.

Moran T.P. (1981)
The command language grammar: a representation for the user interface of interactive computer systems International Journal of Man-Machine Studies, 15, p. 3-50.

Norman D.A. (1981)
The trouble with UNIX Datamation, 27 no. 12, p. 139-150.

Norman D.A. (1983)
Some observations on mental models In: A.L. Stevens, D.Gentner (Eds.), Mental Models. Erlbaum, Hillsdale, N.J.

Veer G.C. van der, Tauber M.J., Waern Y, Muylwijk B. van (1985)
On the interaction between system and user characteristics Behaviour & Information Technology, 4, p. 289-308.

Wilensky R., Arens Y., Chin D. (1984)
Talking to UNIX in English: an overview of UC Communications of the ACM, 27, p. 574-593.

Report from the Association Francaise des Utilisateurs d'Unix (AFUU) on the UNIX community in France.

Phillipe Dax
Philip Peake

1. INTRODUCTION

Writing an entry of this type is not an easy task. But, to keep our editor, happy here goes ....

A history lesson

Firstly, for those of you who haven't heard all this before, a bit of publicity for the AFUU. The AFUU is the French UNIX users' group, and thus a constituent member of the EUUG. We are currently the 2nd largest group in Europe (after the UK ... look out UK, we are catching up fast!).

The history of UNIX in France begins recently. Leaving aside one or two institutions (both commercial and academic) who managed to get UNIX up and running from the Bell Labs' V6 or V7 distribution tapes, the first real appearance of UNIX dates from 1981. The first commercial UNIX distributor in France was Unixsys, who distributed ONYX and PLEXUS systems. Unixsys was rapidly joined by other companies such as Zilog, Tektronix and others. The UNIX system had it's champions in France at this time but remained relatively unknown in the French computing world.

This was the reason for a small group of people to form an association of UNIX users - the Association Francaise des Utilisateurs d'UNIX. This, being almost as much of a mouthful for the French as for anyone else, is normally reduced to AFUU.

The AFUU was created in 1982, and strongly promoted by its first president, Jean-Louis Bernard. The principal aims of the group were to promote (in non-commercial terms) the UNIX system and its derivatives, and to provide a source of information on various aspects of the system, which was hard to come by in France at that time.

In 1983 the AFUU began to grow (95 members), and to structure its activities, with the creation of its first working groups (a group to gather information for the creation of a catalogue of UNIX products available in France, and a group working on standards - headed by Michel Gien), an exhibition of UNIX software and materials at the AGM held at the Prince of Wales hotel in Paris.

In 1984 the growth continued (180 members - a 90% increase). The AFUU continued to function, mainly due to the efforts of certain members, but not with sufficient vigour to satisfy demands in general. A new working group (networking) was formed and driven by Humberto Lucas. As a result of the regular meetings (and some hard work) of this group, the French network (FNET) was formed. During this year, the first newsletter (*bulletin de liason de l'AFUU*) was started by Phillipe Hammes. This was later to become the current AFUU newsletter - TRIBUNIX. The end of 1984 a two day exhibition was organised at the hotel PLM - St. Jacques in Paris, with displays of software, machines and a technical conference.

In 1985 the membership grew to 235 (only a 30% increase this time!). The growth in the association was the cause of two successive moves of the association, eventually ending up at its current location at SUPELEC (Ecole Superior de l'Electricite). The move gave the chance to improve the infrastructure of the organisation, with the employment of a full-time administrator, Anne Garnery.

During 1985 the AFUU collaborated with the EUUG to organise the EUUG-Paris meeting, at the Palais de Congress. The newsletter (TRIBUNIX) was taken over by Phillipe Dax, and given a face-lift. The result is now much more professional, mainly due to the use of T_EX, and a laser printer. A new president, Dominique Maisonneuve, was elected during 1985.

The situation today

The AFUU is somewhat atypical in that it has a large number of commercial members. We are currently looking at ways of increasing the number of end user members. Hopefully, this will happen more or less automatically as the number of installed systems increases.

An area in which the AFUU is very active is that of networking. There exists a French network (FNET), which forms part of EUNET. Until recently the work of providing a backbone for FNET was performed by CNAM, with their machine (vmucnam) forming the gateway into and out of France. The only problem with this arrangement was that CNAM is a higher-education establishment, and so closes during holiday periods. With the increasing load and reliance being placed upon FNET, the decision was taken to transfer backbone activity to INRIA, who can provide a full time service (via their machine inria). One of the most active working groups sponsored by the AFUU is our network group.

The organisation of the AFUU is becoming more established. We have a second full time employee, Jean-Christophe Petithory, who works as a full time technician. His tasks include taking care of running our UNIX machine, which, by the time you read this will hopefully be joined by a second machine; an SM90 on loan from Telmat. The arrival of the SM90, with its increased processing power and disc capacity should allow the AFUU to have a connection to FNET. Jean-Christophe also dedicates part of his time to an advice service for AFUU members, and is available either by telephone or for personal visitors. He will undertake to resolve technical problems, or if these are either large amounts of work, or require specialist knowledge, he can advise on where to find the required skills.

Of course, our full time office manager, Anne Garnery, is also to be found at the same office, which is now located at:

AFUU
SUPELEC
Plateau du Moulon,
91190
Gif sur Yvette

Tel. (+33) (1) 60 19 33 61

We are in the process of deciding the details of our next AGM, to be held in September, either at SUPELEC, or if it looks as if there will be enough interest, and we will run out of space, in Paris. The GGM will, as usual will include a technical session. This session will be free to AFUU members. The sessions themselves will be in French, but anyone interested in attending will, of course, be welcome (details from Anne Garnery).

The following tables describe the actual situation and constitution of the AFUU.

TABLE 1. Division of members

	registered April 86	expected End 86	on-file 85/86
Manufacturers	40	50	60
Public and private institutions	90	115	135
Schools, Universities and Institutes	44	50	56
Individuals	33	42	51
Students	3	3	3
Total :	210	260	305

TABLE 2. Classification of commercial members

	registered April 86	expected End 86	on-file 85/86
Large industrial	10	15	15
Service and training	50	64	80
Nationalised industries	5	5	5
Administration, government	10	13	15
Banking	5	6	6
Other commercial users	10	12	14
Total :	90	115	135

TABLE 3. Classification of academic institutions

	registered April 86	forseen End 86	on-file 85/86
Universities	20	22	22
IUT	4	4	4
Grandes Ecoles	9	10	10
Institutions and Laboratories	11	14	20
Total :	44	50	56

TABLE 4. Geographical distribution

	registered April 86	forseen End 86	on-file 85/86
Paris region	140	178	215
Province (the rest)	65	76	82
Foreign	5	6	8
Total :	210	260	305

TABLE 5. Distribution within France (excluding Paris)

	registered April 86	forseen End 86	on-file 85/86
Alsace	6	7	7
Aquitaine	7	8	8
Bretagne	7	9	10
Centre	2	2	2
Languedoc, Midi-Pyrenees	8	10	12
Nord	3	3	3
Provence-Cote d'Azur	7	9	10
Normandie	8	9	10
Rhone-Alpes	11	13	14
Other regions	6	6	6
Total :	65	76	82

TABLE 6. Evolution of the AFUU

1981 (creation)	8	
1982	45	
1983	95	110%
1984	180	94%
1985	235	30% 150 (April)
1986 (April)	210	40% / April 85

TABLE 7. UNIX machines in France end 1985

Alcatel Thomson (Micromega 32)	3200
Altos (serie x86 et 68000)	1500
Bull Sems (Sps7, Sps9)	700
Unixsys (Plexus, Onyx, Cci)	600
NCR (Tower 32)	400
Divers (DEC, SUN, Apollo, Olivetti)	1600
Total:	8000

EUUG NATIONAL GROUPS

AFUU (France)
c/o SUPELEC,
Plateau du Moulon,
91190 GIF-SUR-YVETTE,
France.

DKUUG (Denmark)
Kabbelejvej 27B,
DK-2700 BRØNSHØJ,
Denmark.

EUUG-S (Sweden)
NCR Svenska AB,
Box 4204,
17104 SOLNA,
Sweden.

FUUG (Finland)
OY Penetron ab,
Box 21,
02171 ESPOO,
Finland.

GUUG (Germany)
Mozartstrasse 3,
D-8000 MUNICH 2,
Federal Republic of Germany.

IUUG (Ireland)
Glockenspiel Ltd.,
19 Belvedere Place,
DUBLIN 3,
Ireland.

i2u (Italy)
Viale Monza, 347,
20126, MILANO,
Italy.

NLUUG (Netherlands)
Xirion bv,
World Trade Center,
Strawinskylaan 1135,
1077 XX,
AMSTERDAM,
The Netherlands.

NUUG (Norway)
Unisoft AS,
Enebakkvn 154,
N-0680 OSLO 6,
Norway.

UKUUG (United Kingdom)
Owles Hall,
Buntingford,
Herts. SG9 9PL,
United Kingdom.

UNIGS (Switzerland)
c/o Instutut fur Informatik,
ETH Zentrum,
8092 ZURICH,
Switzerland.

UUGA (Austria)
TU Wien,
Inst fur Praktische Informatik,
Gusshausstr 30/180,
A-1040 WIEN,
Austria.

EUUG

The Secretariat: European UNIX® systems User Group, Owles Hall, Buntingford, Herts SG9 9PL, UK. Tel: Royston +44 (0) 763 73039 Facs: Royston +44 (0) 763 73255
Network address: euug@inset.uucp

Digital VAX Product Announcement
A Personal Evaluation

Boyd Roberts

The Instruction Set
now at
Austec Software

1. OVERVIEW

I attended what DEC claimed to be the "Information Technology Event of 1986" at the Novotel in London. DEC were launching their new VAXen, a new VAX bus and for good measure the odd 4GL. Prior to the event DEC were making wild and wonderful claims, facts notwithstanding.

2. CHAMPAGNE SUITE AT THE NOVOTEL

After an interminable wait at the cloakroom I proceeded to the Champagne Suite hoping to find some of the French stuff, but only to find some truly foul coffee. There were between 500 and 800 attendees including a large number of badge wielding DEC personel. Not far from the coffee were some new DEC workstations, or VAXstations to the initiated.

There were four VAXstations in total, looking suspiciously like SUN III's save for the colour displays. Two ran ULTRIX and two ran VMS. The DEC ULTRIX support guy said that ULTRIX was "fully System V compatible". Except, they didn't support some of the more dubious System V *enhancements* (shared memory, messages and semaphores). Not really "System V compatible" I thought. When asked about a network file-system for these systems he commented that yes they had it, and yes it was developed at DEC. A very likely story.

The ULTRIX VAXstations were bearable, but the microVMS VAXstations left you wondering why they bothered. The mouse interface was truly naive. The windowing was slow and the pop-up VT100 windows were inconceivable. You selected at VT100 window, up it popped, complete with the message

```
Username:
```

This floored me. You had to log in each time you *swept* out a new window.

They also had menu selections of various bit-mapped terminal standards (a clock, bouncing balls etc). All were very slow.

3. *THE PRODUCT LAUNCH*

This is where the hype started. A large video screen spewed forth shots of various high technology processes mixed with various shots of "digital" and "solutions". Not only that, the room pulsated with mechanistic 80's muzak: drum machines and synths all doing their worst. The managing director of DEC UK got up and told us all how DEC were **the** computer company, putting your needs first, etc, etc.

Various DEC marketing types were to follow stressing that DEC stood for compatibility and were leaders in computer networking. The networking aspect I found a bit hard to swallow. Flogging a lot of DECnet ethernets does not equal networking. As for compatibility; they were doing it with hardware when you've got to do it with software.

Then the lasers started. Was this really the Hippodrome?

3.1 *VAX 8200/8300*

Here were the new VAXen. They are the replacements for the 750 and the 780. Stressing that they were smaller, faster and less expensive than their predecessors. Well, what could you say? They were just better packaged 70's technology. The 8200 claimed to be 15% faster and £70k cheaper than the 780 (but these were list prices). You could pick up a minimal configuration 8200 (9 track tape + RA81) for £120k in mid '86.

The 8300 is the big sister to the 8200. It's claimed to be twice as fast as a 780, but it'll set you back £160k. Again, that's list price for a minimal configuration. Both the 8200 and 8300 come with the new VAXBI (which is like the 780's SBI). Attached to that are the CPU and the the VAXBI adapters. However, here's the catch. The 8200 has one CPU, the 8300 has two.

Here we come face to face with the "DEC MIP". DEC would have you believe that CPU power stacks up like Lego blocks. We know it doesn't. However, if you can exploit the parallelism you do get a win. Is the 8300 a 2 MIP machine or a machine with two 1.15 MIP CPUs?

3.2 *VAXBI*

This is DEC's new bus. They claim that it has "faster throughput", but at 5Mb/s it's slower than the 780's SBI (13Mb/s). They quoted the 5Mb/s, but the technical summary claims that it's 13Mb/s for 16 byte transfers. Obviously it's slower for smaller transfers. My "red" book says the SBI does 13Mb/s for 4 byte transfers. So, the SBI is still faster. VAXBI "faster thoughput"?

As they were harping on about reliability, they continued in this vein by commenting that the VAXBI has "full error checking". Well, you'd certainly hope so.

3.3 VAX 8800

By the time the 8800 came around I was beginning to get a bit bored with the "VAX in any flavour" approach. However, I was to be suprised. Till now the VAX was not exactly a fast machine. The 8800 is 11 DEC MIPs. It has 2 VAXBIs as well as a *fast* memory bus.

The 2 VAXBI hang off the memory bus, as do the CPU and the memory. This *fast* bus is. It's claimed to be able to do 60Mb/s. That was a bit more like it. In all of these presentations they seem to forget that most peripherals can't saturate the bus, far from it. The memory is, of course, another issue.

A minimal configuration 8800 will set you back £608k. Sound ok? Guess again. It's got 2 CPU's. My feeling is that these you get 2 5 MIP CPUs. Again those DEC MIPs raise their ugly heads.

You can put 32Mb of memory in an 8800. Memory comes in 4Mb chunks and these chunks are made up of 256k chips. Also, for your £608k, you get an ethernet and a clusterbus (is that the term?). A *cluster* allows you to chain CPUs together to share peripherals and CPUs.

3.4 Clustering

The subject of *clustering* was hailed as a way to get your DEC MIPs to stack up. You get networking for free via this approach (so they said). A lot of nice diagrams showed how you could *cluster* CPUs and network microVAXen and terminal concentrators, supposedly adding up to a tens (or hundreds) of MIPs (DEC ones). The mud began to fly at this stage. DEC started to compare their *clustered* approach to various other vendors' mainframes. The *clustered* DEC VAXen versus the IBM 3083 etc. I was a bit dismayed at this, but the *clustering* is sort of cute. Would the gullible industry types fall for these slickly packaged misconceptions?

3.5 4GL

DEC have finally got on the 4GL bandwagon. I know that Burroughs have been doing this for years. DEC give you a mouse-based bitmapped interface to what they call a "COBOL generator". You draw up these dinky little data-flow diagrams with icons and out spews 10,000 lines of COBOL. The increases in productivity were done to death and we were informed that all our applications programmers were behind schedule.

Apart from the applications programmer interface they also provide an interface for managers. The manager can use this menu driven query language to do database inquiries, extended to the point where the retrieved data can be turned into graphs etc. With all this lot they've given you layered databasing.

4. Cocktails

I was quite thirsty by this stage. I headed for the free booze. What a poor show that was. Only beer, wine and soft drinks. Not even a G&T. And, there was no food. After the odd lager I was prodded to give the top UK ULTRIX guy a hard time. I did my best, but he knew his stuff and was not easily ruffled.

The Secretary
European UNIX® systems User Group
Owles Hall
Buntingford
Herts. SG9 9PL.
Tel: Royston (0763) 73039.