A model of a self-organising data management system

P. Dearnley

School of Computing Studies, Computing University of East Anglia, Norwich NOR 88C

The concept of a self-organising data management system, presented in an earlier paper by Stocker and Dearnley, is summarised and a model of such a system is described.

(Received September 1972)

1. The self-organising data management system

The self-organising data management system is designed to serve the interests of a variety of users, possibly remote and uncoordinated, whose usage of the database either cannot be completely determined at the system design stage or will change as the database and the users' interests develop. In such a system the structuring of files takes place at access time (or as folio management, see Section 6) and not at data load time, which results in a slower response time but is in line with the attempt to minimise overall cost. A further feature of the system is that duplication or partial duplication of the same data in files of different structures and generations is used to balance access costs against storage costs.

The system is based on the following principles:

1. The system has the capability to determine and implement suitable structures for its files. Structures are chosen with the object of minimising the total cost of known or predicted accesses.
2. The access strategy adopted is constructed by the system.
3. Any correctly specified access can be made by adopting some stategy and the user is given a cost quotation in advance.
4. The system can restructure or update files as a result of (*a*) an accepted cost quotation or (*b*) by observing patterns of usage and finding a different structure economically advantageous to the body of users as distinct from an individual.
5. The user is allowed to leave requests in the system in the hope that batching or structural changes will eventually reduce the cost of his task to an acceptable level.

The model described in this paper implements principles 1 to 4 in some form and is used to demonstrate that a viable database management system can be built along these lines. A 'working model' of a self-organising data management system was chosen in preference to a simulation of the file access (after the fashion of Senko, Lum and Owens, 1965; and Lum, Ling and Senko, 1970), on the grounds that the value of real operational experience would outweight the time and effort saved by simulation.

The original concept of a self-organising data management system is described in more detail in Stocker and Dearnley, 1973.

2. Users' requests

The user views the database as a number of distinct packets of data and he is unaware of the duplication, partial duplication or restructuring which may have taken place since the data was inserted in the database. This packet of data is referred to as a *folio* to distinguish it from the files which may now represent the data in various forms. The user specifies his request in terms of a folio and if any knowledge of particular files is ever required then this is given by the system. Simple operations on folios (such as folio definition, deletion, input, etc.) can be carried out without any decision-making on the part of the system; these operations are invoked immediately by the request interpreter (see **Fig. 1**). More complex actions (which will include searching, sorting, indexing, updating, etc.) require some use of the decision-making capabilities of the system and are passed to the 'route finder'. This route finder is responsible not only for choosing methods but also supplying the user with a cost quotation. Only if this quotation is accepted is the chosen method implemented by the 'complex action interpreter'. Thus the user views the system in two ways. In a direct manner for simple operations and in a 'bidding' manner for more complex actions where a request may be withdrawn if the cost is too high.

The model is built with an overall structure following **Fig. 2**. Any suitable application module can be constructed to drive the data management module. The application module used in the model is a simple form of information retrieval. The user requests take the form of a file of key values to be found in a particular folio. The records found are placed in a file which can be used for subsequent retrieval, processing or printing.

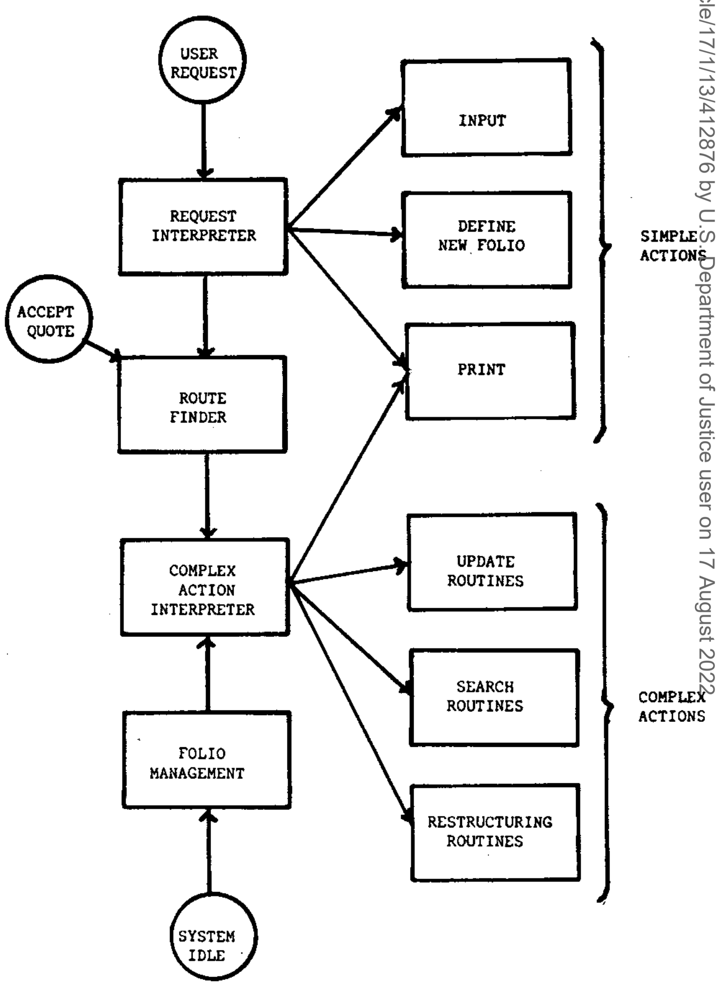
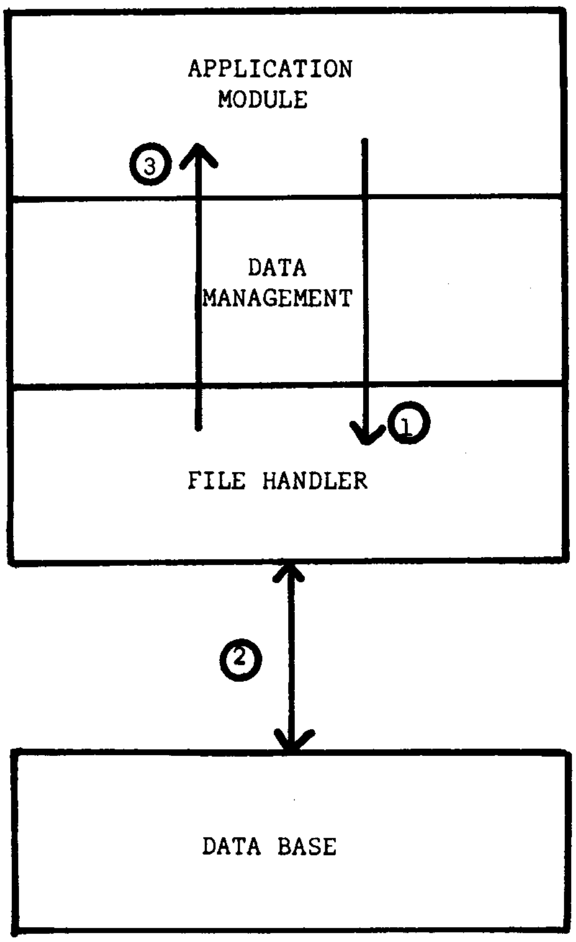


Fig. 1. System block diagram



(1) ACCESS REQUEST

(2) FILE ACCESS

(3) RECORD RETURNED FOR PROCESSING

Fig. 2. Overall structure

3. System file structures and access methods

Item 1 of Section 1 indicated that the system has a variety of file structures at its disposal. Those structures chosen for inclusion in the model represent the range supported by an operating system such as OS/360 (IBM 1966; IBM 1967) or common house-keeping packages (ICL 1968). Thus the file structures which the system can choose are similar to those which a system designer might readily select without extra programming support. However extra data management methods (such as those of Lefkovitz, 1969) could be included by adding additional modules. This would represent the use of a specially programmed data management section in a normal system design.

The file structures supported by the system (following the terminology of Clifton, 1969) are

1. Serial file with records in consecutive locations but not in key sequence.
2. Sequential file, as 1 but ordered on one or more record keys.
3. Indexed sequential, as 2, but with an additional file forming an index to the highest key value in each physical section of the file.
4. Random, with records located by a key-to-address transformation function.

The system has modules to perform the appropriate conversion of one structure into another. These modules include sorting, indexing and key-to-address transformation. In addition sub-records containing selected fields can be extracted from files. To update files the system has modules to append records to serial files, to merge sorted updates with sequential files and to insert records in random files.

To gain access to records the system has the following access methods:

(*a*) a serial search of any file type.
(*b*) a binary search of file types 2 and 3 above.
(*c*) a merge search of file types 2 and 3.
(*d*) a selective search, using indexes, of file type 3.
(*e*) a selective search, using key-to-address transformation, of file type 4.

4. The system in operation

The system starts with an empty database, and then various folios are defined and input. These folios are used to satisfy users' requests; in response to some (or all) individual requests, various file structures are implemented and duplicates or partial duplicates of the files created. To satisfy subsequent requests the system will have a choice of access methods applied to a number of files.

However a stimulus other than users' requests can cause the system to operate. This stimulus is 'system idle' (see Fig. 1). When the system is idle it reviews the usage made of the database and can cause new files to be created or old files to be restructured to make the predicted usage of the database more economic for the body of users rather than any one individual request. The folios defined are recorded in the system directory as folio header records and this definition is referred to by the user when making requests. As the system operates new files are formed and each has a file header record created by the system and 'owned' by a particular folio (see **Fig. 3**). As particular versions of a folio are used in searches so additional records called 'search addenda' are appended to the system directory.

5. Route finding

From the previous sections we know that the system chooses various file structures and access methods, how are these choices made? The choices made in response to a user request are referred to as a *route* and the activity undertaken to make the choice *route finding*.

A folio is viewed as a directed graph. Each node represents a field in the folio. Each arc represents the access of a non-key field from a key field facilitated by a particular file's structure. The weight of an arc represents the expected cost of using that file to obtain a value of a particular field.

The first phase in route finding consists of finding a subgraph which joins all the fields whose values the user specifies to all the fields whose values the user wishes to retrieve. The subgraph chosen is that with the minimum total expected cost. The chosen subgraph represents the use of a particular set of files to satisfy the user request. This set of files is chosen from existing files only. The second phase is to examine the graph and see if any particular set of arcs could be drawn to produce a cheaper subgraph. If a particular set of arcs can be drawn then this represents making a new file from some existing file and accessing this new file. This phase is called *route modification* and the total cost includes the cost of constructing the file in addition to the cost of using the new file. When a route is chosen the generations of the folio covered by the files used are recorded. The range of generations covered and the cost are offered to the user. If the user wishes to have a wider cover of generations the route finder is re-entered with a flag set which ensures that the correct range of generations is covered. Thus the user has the option of a cheaper access restricted in gener-

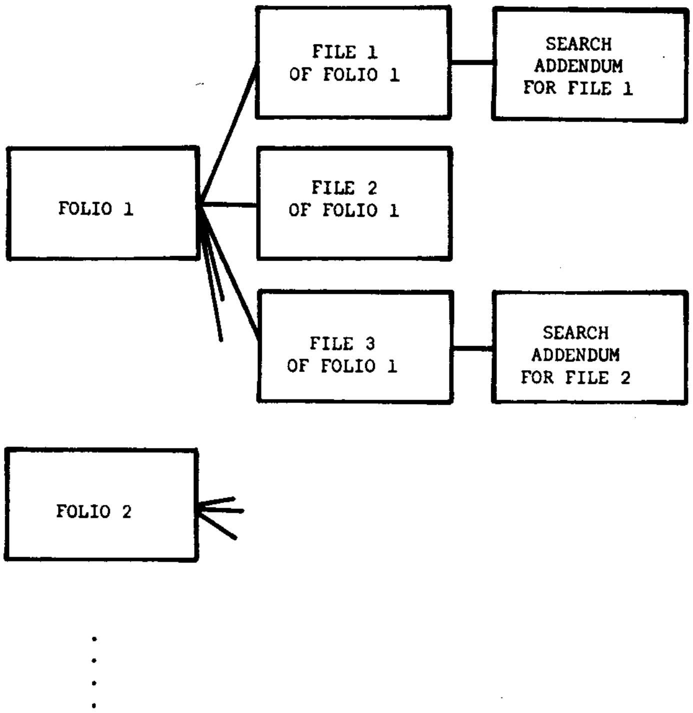


Fig. 3. System directory structure

ation coverage if this is feasible. If a particular generation range is required by the user then the route finder may use files of updates in addition to the normal versions of the folio or apply these updates to one or more files.

6. Folio management

The creation of new files or the restructuring of old files for the overall benefit of the users rather than as a result of one request is referred to as *folio management*.

Such changes in the structure of the database occur following the system idle stimulus. When the system is idle a review mechanism is set in motion. This review mechanism examines the search statistics appended to the system directory. The use of the fields within a folio is examined and the 'best' files to meet this usage compared with the actual versions in the folio. If some dominant trend is observed which is not well catered for in existing versions then a new version is produced. This ensures that the folio changes as its usage changes even though no one request justified such a change. This review mechanism assumes that the trend of recent searches is likely to be maintained in future usage. The review mechanism does not delete any versions; this is, at present, only done by direct intervention from outside the model system. When a review is complete a set of actions has been selected; these actions are then passed to the complex action interpreter in the same fashion as a route from the route finder.

7. Costing

For both route finding and folio management, costing information is required. The costs of serial searches, merge searches, sorts, indexing, extracting files of subrecords, etc. are determined by file and record length. The cost of binary and selective searches require a knowledge of the number of records with any given key value (for at a given address determined by a key-to-address transformation), and thus the expected search length to satisfy any one access making up part of a request. The distribution of key values is recorded in the system directory when files are structured and thus an expected cost can be obtained. For example, as the records written by the output phase of the sort are placed in the output file 'control breaks' on the major sort key are recorded; this gives the expected number of duplicate records for any one key value and thus the expected search length.

8. Action implementation by the system

User's console input messages are decoded by the request interpreter (see Fig. 1), and divided into simple requests and complex requests (see Section 2). The complex request can, if a quote is accepted, give rise to a queue of actions representing a route. This queue can also be loaded from the folio management module. The queue is interpreted by the complex action interpreter which can call upon the appropriate series of simple and complex actions in a sequence dictated by the order in the queue. Each complex action involves a search, update, restructuring or file housekeeping routine. All intermediate results are stored in temporary files which are erased at the end of the route. Each routine calls on a lower level routine for basic file handling functions. The file handler functions supplied determine the choice of access methods and further functions can be added. The file handler calls upon a software paging system which uses 512 character pages. Each file is held on a set of pages specified by a linked list. This list is kept as part of the system map and is held in core during file processing. As files are handled two classes of statistics are recorded. At the paging level the date, time and frequency of usage of each page is recorded and stored on the page. When a file is being searched the search statistics are accumulated in core store. The search statistics give the fields used to access the file, the fields retrieved and the most commonly used pages. These search statistics are appended to the system directory at the end of the search and used in folio management.

9. Program implementation

The model system described in this paper has been implemented on the ICL 1905E. The model is concerned with testing the viability of the self-organising data management system concept and not with operating efficiency; this has meant that where possible existing software and high level languages have been

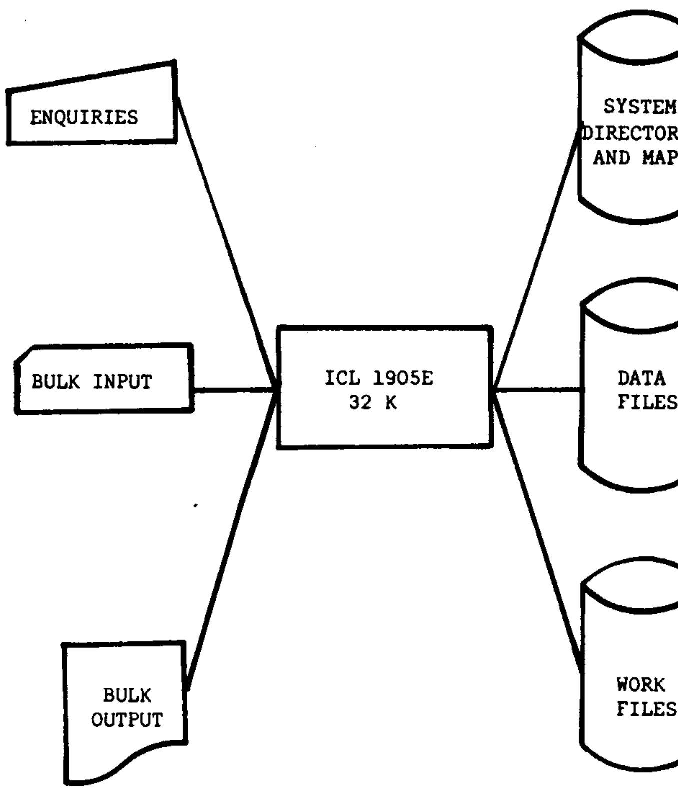


Fig. 4. Machine configuration

used. The on-line environment is provided by MAXIMOP and ICL sorting software is incorporated in the program. The low level routines are written in the ICL PLAN assembly language and other routines in 1900 FORTRAN. The program uses between 18K and 32K words of core store.

The larger core sizes are used to hold more of the frequently used areas permanently in store and thus reduce disc traffic. The database is restricted to one million characters with a further one million for 'back-up' purposes. The database can be divided into six distinct folios and may have up to 30 files. The overall machine configuration is shown in **Fig. 4.**

The operation of the model is monitored by an (optional) trace mechanism which records each decision and action taken. The trace information is off-lined and printed if required for demonstration or debug purposes.

10. Testing experience

A database has been set up using only one folio and a test case request prepared. The test case was tried with the folio with one file only and the system chose a serial search. The size of the test case was increased and the system chose to sort and index the one file and perform a selective indexed search. Along series of short requests were then made, interspersed with idle time. None of these short requests themselves occasioned any changes but new file versions were produced during folio management. These new file versions had shorter record lengths and were sorted and indexed. The original test case was rerun and the route chosen used two short intermediate files before completing the request in a similar fashion to the example in Section 7 of Stocker and Dearnley, 1973.

The test folio has eight attributes in field $a_1, a_2, \ldots, a_8$. After the series of short tests the following files existed:

File A containing attributes a_5, a_8,	sorted and index on a_5
File B containing attributes a_2, a_5,	sorted and index on a_2
File C containing attributes a_1, a_3, a_4	sorted and index on a_1
File D containing attributes a_1, a_2, a_4	sorted and index on a_1
File E containing attributes a_1, a_2, a_3, a_4, a_8	sorted and index on a_8
File F containing attributes $a_1, \ldots, a_8$	not sorted.

The test case was:

given values for a_1 and a_2
find values for a_3 and a_4

When this test case was rerun the following route was followed:

(*a*) selective search by index of file B using a_2 for a_5
(*b*) selective search by index of file A using a_5 for a_8
(*c*) selective search by index of file E using a_8, a_1, a_2 for a_3, a_4.

This route was chosen in preference to the original serial search of file F and in preference to a selective search by index of E on key a_1. Key a_1 had few distinct values (and thus long search paths) compared to key a_2, which was unique.

In general, indexed sequential files have been little more costly to construct and keep than sequential files and are usually preferred for subsequent processing. Further, the random file offers little over the indexed sequential and presents problems in the selection of key-to-address transformation functions.

11. Conclusions

The behaviour exhibited by the model has shown that a data management system can be constructed which will

(*a*) choose suitable file structures
(*b*) construct access strategies
(*c*) attempt to minimise the cost of access
(*d*) change the database according to data usage.

The effort required to construct the model and the machine resources used indicate that implementation and use of a full scale system is a viable proposition.

Further, this model has helped to pinpoint areas requiring further work with a view to implementing a full scale system. These areas include sophisticated route finding, the value of differing versions of files, the automatic construction of key-to-address functions, the costing of complex access methods and the prediction of trends of future usage.

Corrigendum*

Corrigendum to the paper *Self-organising Data Management Systems* published in Volume 16, Number 2.

The right-hand column of page 102 contains a table of file attributes. The line 15 lines from the bottom should read

'File *D* containing attributes a_2, a_3, a_4 sorted on a_2'

and the line 14 from the bottom
'File *E* containing attributes a_1, a_2, a_3, a_4, a_8 sorted on a_8'

References

CLIFTON, H. D. (1969). *Systems Analysis for Business Data Processing.*
IBM (1966). *Introduction to IBM System/360: Direct Access Storage Devices and Organisation Methods.*
IBM (1969). *IBM System/360: Operating System Supervisor and Data Management Services.*
ICL (1968). *Direct Access Manual No.* 4107.
LEFKOVITZ, D. (1969). *File Structures for Online Systems.*
LUM, V. Y., LING, H. I., and SENKO, M. E. (1970). Analysis of a complex data management access method by simulation modelling. *AFIPS Fall Joint Conference*, 1970.
SENKO, M. E., LUM, V. Y., and OWENS, P. J. (1968). A file organisation evaluation model (FOREM). *IFIP Congress* 1968.
STOCKER, P. M., and DEARNLEY, P. A. (1973). *Self organising data management systems. The Computer Journal*, Vol. 16, No. 2, pp. 100–105.

*See also the letter by F. Poole on page 95.