Network Working Group
Request for Comments: 195
NIC 7140
Categories: D.4, D.7

G. H. Mealy
HARV

16 July, 1971

Data Computers -- Data Descriptions and Access Language

According to the minutes of the NWG meeting in May (RFC 164), it appears that a unified approach to Network data management is being proposed to CCA. The purpose of this paper is to discuss some of the problems involved and to suggest possible avenues of approach toward their resolution. Parenthetically, I believe that a non-unified approach leads to even worse problems.

My main remarks are predicated on a few assumptions and their consequences. Since some or all may turn out to be wrong, it seems appropriate to state them explicitly. The steps in the arguments leading from the assumptions to their consequences may appear to be (and in fact may be) less than obvious. They are all of a piece, however, and revolve around the necessity for doing business with a number of dissimilar HOST systems while attempting to make it unnecessary for an individual user or user program to know the details of data file organization and representation. Given this as an objective, I believe that the arguments are quite direct.

Assumptions

1. We face the usual set of naming, cataloging, protection, backup, etc. problems.

   (I say this only to dismiss the subject as far as the following is concerned. In my estimation, these problems and feasible solutions are reasonably well understood; our real problem in this area is in reaching agreement on specifics while leaving sufficient ratholes for future expansion).

2. Files stored will contain arbitrarily complex data objects.

3. The organization of any file (that is, the way its structure is mapped into physical storage by the data computer) will normally be unknown by the user.

4. Data items in files may be stored in arbitrary representations (e.g., those of the originating user's HOST rather than that of the data computer or other "standard" representation).

5. Access to a file will normally be to some subset of it. (I.e., the unit for transmission will usually be part of a file rather than the whole file, and access will not necessarily be sequential).

Consequences

1. A method of data description significantly more powerful than now commonly available (as with COBOL or PL/I) is required. The descriptions must be stored with the files. Data item representations and storage organizations must be describable.

2. The data computer must offer a "data reconfiguration service", based on use of the data descriptions.

3. A representation and organization-independent level of discourse must be made available for controlling access.

Data Description

As it happens, the descriptive facilities in ELl (References 1 and 2) are almost adequate as they stand. ELl is an extensible language -- the compiler and interpreter for ELl are principal components of a system implemented on the PDP-lO at Harvard -- which allows the definition of arbitrary data structures in terms of a few primitive data types (BOOL, CHAR, INT, REAL, SYMBOL, MODE, FORM, and ROUTINE). These data types are of the sort I called "generic" in Reference 3. To the EL1 implementation on the PDP-10, say, we would have to add methods to describe a specific representation of INT, etc. and primitive routines to convert between specific representations.

In the ECL system (in which EL1 is embedded), there is no rigid distinction between compile time and run time. In particular, if the arguments and free variables of a routine are evaluable at compile time, then the routine is evaluated and the value replaces the call. More generally, arbitrarily large amounts of a routine being compiled may collapse into values. As far as the data computer is concerned, this offers the possibility of producing tailor-made data reconfiguration programs, taking maximum advantage of the data descriptions at compile time rather than using a strictly interpretative mode of operation.

Access Language

Here, I am on less firm ground. I will suggest, however, that some of the ideas of Sattley, et al (Reference 4) deserve consideration. I will quote from the Reference:

"... Our proposal is a language for describing the transferable features of files, in which conventional programming languages (e.g., FORTRAN, ALGOL, etc.,) can be embedded, and from which the information necessary to optimize the use of secondary storage can be easily abstracted. This language defines our abstract model of secondary storage in the same way that FORTRAN defined an abstract machine. This language should have (at least) the following features:

1. File declarations name the file and the elements in the file, and specify the range of forms that the elements can take. Each file has precisely one named element. Each file includes the (maximum) size (in number of elements) of the file.

2. Subsets of files can be created by means of grouping declarations. Such subsets can be nested.

3. Subsets of files can be named by means of naming declarations. Such declarations can also name individual elements of the file. Some form of implicit naming, allowing language constructs such as GET ANOTHER TRIPLE, is included.

4. Members of a set (i.e., elements in a subset or file, subsets in a containing subset or file) can be ordered by order declarations. Some form of arbitrary but fixed ordering, allowing language constructs such as GET NEXT TRIPLE, is included.

5. The portions of a file transacted with at a point of access is declared. The size of this portion can be expressed in absolute or relative terms.

6. At each point of access to secondary storage, an environment is described (or referenced) which contains those declarations of types (l)-(5) necessary to define the transaction with secondary.

A language with the above features makes no mention of hardware devices, but it provides the programmer with the means of defining the algorithm-dependent features of his files so that those files might be transferred efficiently from machine to machine".

In the Sattley, et al study, the notion was that a compiler would take the source program and actually compile the hardware-dependent file accessing code. In our environment, we are concerned with control commands to the data computer (e.g., GET NEXT WALDO) and supplying the data computer with enough information to define a WALDO. The basic functions would seem to be the same, in either case, albeit implemented rather differently.

References

1. Wegbreit, B. The Treatment of Data Types in EL1. Technical Report, Division of Engineering and Applied Physics, Harvard University, Cambridge, Massachusetts, May 1971.

2. Wegbreit, B. The ECL Programming System. Technical Report, Division of Engineering and Applied Physics, Harvard University, Cambridge, Massachusetts, April 1971.

3. Mealy, G. H. Another Look at Data. AFIPS Conference Proceedings, vol. 31, 1967 Fall Joint Computer Conference

4. Sattley, K., Millstein, R. and Warshall, S. On Program Transferability. Report CA-7011-2411, Massachusetts Computer Associates, Wakefield, Massachusetts, Movember 1970.