Lawrence Berkeley National Laboratory
LBL Publications

Title
Data Structure Charting Methods

Permalink
https://escholarship.org/uc/item/5t72q5gx

Authors
Hart, Esmond
Ringland, Gill

Publication Date
1975

LBL-3659 Rev.
c. 1

DATA STRUCTURE CHARTING METHODS

Esmond Hart and Gill Ringland

January 31, 1975
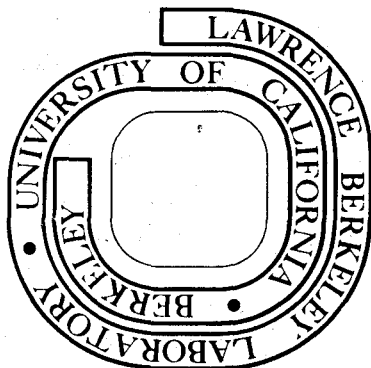
LBL-3659 Rev.
c. 1

DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.

LBL-3659 Revised.

DATA STRUCTURE CHARTING METHODS

Esmond Hart

Computer Analysts and Programmers Ltd.,
14-15 Gt. James St.
London WC1, England

and

Gill Ringland* †

Lawrence Berkeley Laboratory
University of California
Berkeley, California 94720

*On Leave of Absence from CAP, October 1974 to October 1975.

ABSTRACT

The growing importance of data base implementations, and the

consequent need for describing interrelations between data elements,

has highlighted the lack of tools for describing data structures.

The paper describes with examples, a charting method which has

been used successfully on a number of large projects. The important

features are: the compactness of charting methods, which enables large

and complex systems to be understood; and the ability to represent also

detailed information such as the structure of data sets.

DATA STRUCTURE CHARTING METHODS
by
Esmond Hart, Computer Analysts and Programmers Ltd.

and

Gill Ringland, Lawrence Berkeley Laboratory*

INTRODUCTION

'The crucial role of correct analysis and design of a computer system, in providing a cost effective hardware and software solution to a problem, has long been acknowledged. More recently, computer professionals have become aware of the need to satisfy other conditions -- such as providing a reliable system, or one that is easily maintained, or one that contains 'unbreakable' security facilities. In general, it can be said that computer professionals must now satisfy a wider audience of the viability of their product, before it enters the user arena.

Software professionals are especially vulnerable, in that they have comparative paucity of tools with which to analyze or describe their products. Hardware designers, for instance, can communicate via circuit diagrams which use internationally recognized symbols. These diagrams enable an entire configuration to be represented in a way which expresses concisely and accurately a large amount of information. The diagrams can be used to analyze the interactions of components, or to discuss the overall design with a manager or auditor.

In computer software, the only widely recognized form of system representation is the flow chart. This was developed originally for program logic, and has been extended to cover system flow. For instance, a set of charting conventions for use in designing interactive, teleprocessing systems, has previously been described (1). However, these charts, analogously to program flow charts, mainly represent the execution sequence within the software, and only marginally represent the software architecture. The software architecture, both the interrelation and control structure of program modules, and the interrelation and structure of files, data areas, and control blocks, is usually complex in large systems. Certainly in data base or teleprocessing systems, the senior designers concentrate in the main on defining these interrelations. Very often, the specification and production of individual program modules in a well designed system is within the capabilities of relatively junior staff. These staff work within the system architecture defined by the design team.

We, in CAP, have needed, because of the size and complexity of some projects in particular, to develop an effective methodology for expressing this system architecture. The methodology is defined in (2): the present paper is intended to illustrate the utility of the charting method for the particular problem of representing data structure.

---

*On leave of absence from CAP, October 1974 to October 1975.

The problem is discussed under four headings. After establishing the terminology we will use in referring to groupings of data, we discuss the representation of data attributes within the charting methodology. Secondly, we describe how logical relationships -- such as inclusion or exclusion -- can be charted. Third, we discuss representation of linkage mechanisms. The fourth (and major) section shows the use of the charting symbols to describe data structure. This section covers control and storage/access structure, and also the relationship between data within the same and separate structures.

The methodology encompasses the representation, concisely and un-ambiguously, of a wide range of constructs. It is, therefore, necessarily complex. Additional examples and case studies are included in (3), which has been used as a teaching manual for programmers.

TERMINOLOGY

The aim of this paper is to describe a methodology for describing relationships between data items. Since we realize that there is by no means unanimous consensus on terminology in this field, we define below the sense in which we use the relevant terms.

Data element: The basic unit of information to be defined (for example, "address" or "line in use indicater").

Data item: A logical grouping of data elements, such as a control block, personnel record, or program module.

Data structure: A collection of data items with some attribute in common, stored and accessed in some manner, (for example, the personnel records stored in a table, or a last-in-first-out-queue, etc.).

Set of data structures: A collection of data structures with function-ally equivalent contents, for instance all the tables containing personnel records.

It should be stressed that, while data items are essentially logi-cal groupings of elements, data structures are organizations. Thus, the storage and access algorithms for handling the data items are an es-sential part of the structure definition.

The aspects of data architecture that a systems designer would normally consider are:

Attributes: Data set attributes may describe the organization of indi-vidual items in the set, or refer to the set as a whole (for example, "location");

Logical relationships: The commonality or otherwise of data items stored in different structures;

Linkage: The duration and control/slave nature of links between items or sets;

Structure: The storage and access method, such as queue, table, etc., of a group of items, and the relationship between items in the group.

ATTRIBUTES

The attribute information referring to the individual data items can be summarized as follows:

Number of items in the structure;

Number of structures in the set;

Key which uniquely defines each item; or

Tag which orders the items on non-unique keys.

The information about the set as a whole includes:

Name of the set;

Location, i.e., where the set resides;

Availability, i.e., when items join or leave the set;

as well as the structure, linkage, and logical information.

The set is represented within the charting methodology by a rectangle divided into a header and body (see Figure 1). The item-oriented attributes are written in the body of the rectangle, or "chart element". The set-oriented attributes are noted in the positions shown outside the element.
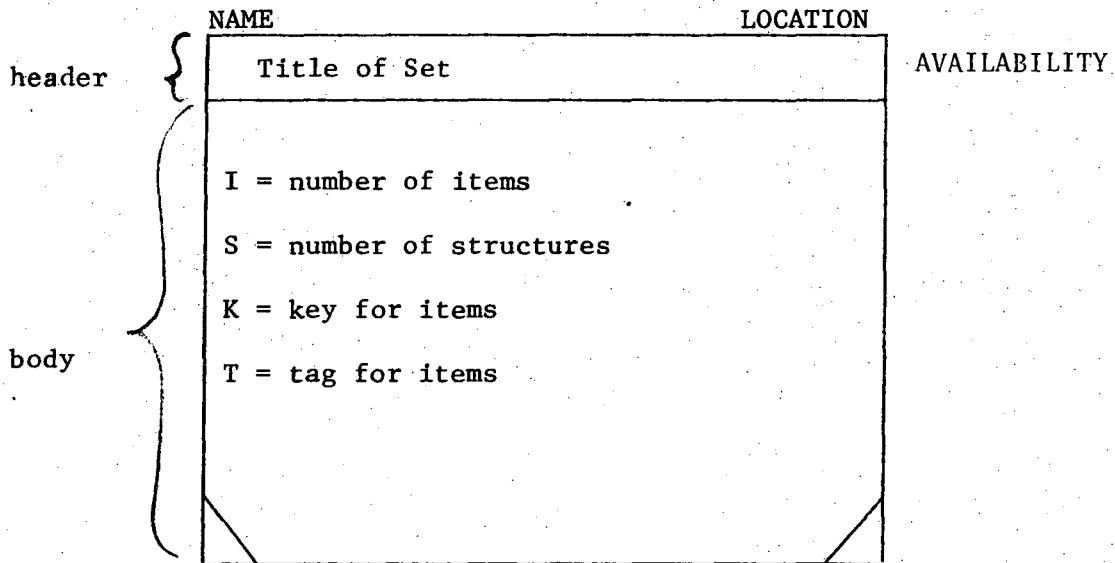
Although in this paper we are not primarily concerned with programs, it is relevant to note here that a similar chart element, but with cut off lower corners, is used to represent programs.

Attributes may be fixed for the duration of the life of the item or set, or may be updatable. The convention used for this is that:

- precedes a fixed value attribute,

* precedes an updatable attribute.

LOGICAL RELATIONSHIPS

In this section, we define the charting symbols used to represent inclusion and exclusion relationships between sets. The conventional graphical representation of set relationships is by means of Venn diagrams (4). These are difficult to incorporate into any scheme showing additional information since they rely on juxtaposition and shading.

```
                NAME                              LOCATION
header  {      ┌──────────────────────────────────────┐
               │   Title of Set                        │        AVAILABILITY
               ├──────────────────────────────────────┤
               │                                        │
               │   I = number of items                  │
        {      │                                        │
body           │   S = number of structures             │
               │                                        │
               │   K = key for items                    │
               │                                        │
               │   T = tag for items                    │
        {      │                                        │
               │ ╲                                    ╱ │
               └──────────────────────────────────────┘
```

corners cut off to represent a program

Figure 1. Set Attributes

Figure 2, therefore, shows set relationships within the data architecture charting scheme. Heavy dotted lines are used to connect two or more chart elements representing sets. The figure also shows fixed attributes "type" of the data items X and Y, members of sets B and C, respectively.

LINKAGE

An important characteristic of data is that a data item may contain information which can serve to uniquely select an associated data item in its own or another set. Such information is a special attribute of the item termed a link. Figure 3 summarizes the types of link distinguished within the methodology.

The interpretation of the solid link is that the item always contains a link to the specified item. A broken line is used to represent an optional linkage. This means that it is not always true that each item X contains a link to some item Y. (This may lead to an implementation problem in recognizing, for example, that a pointer field in a set A item has the 'null' value and does not actually address any set B item.)
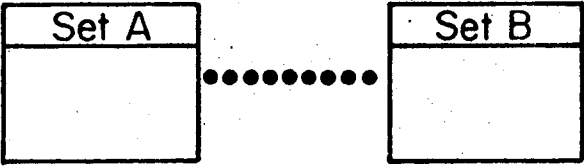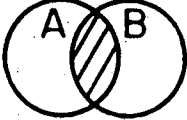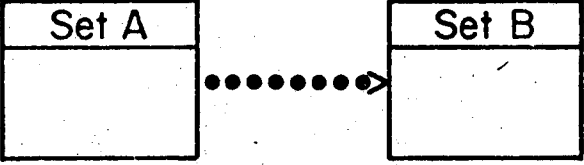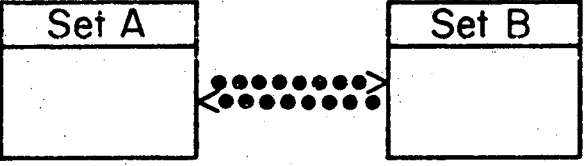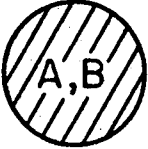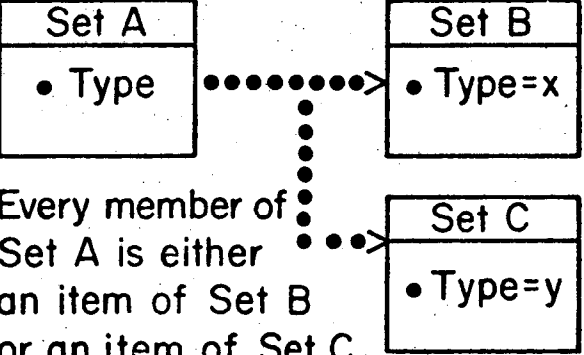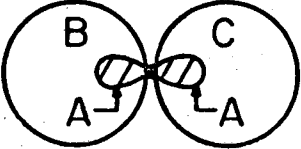
The charting method provides no special symbols for specifying exactly how a linkage is affected. Whether the link is a fullword or halfword pointer or a disk address is considered to be an implementation detail and unimportant as far as an overview of data architecture is concerned. If details of the linkage implementation need to be shown in a chart, comments can be written on the arrows as shown.

A link is fixed when it remains unchanged throughout the 'life' of the item containing it. This means more precisely that the link is created when the containing item is stored and thereafter remains unchanged. The notation for a fixed link is a point above the stem of the arrow. When the point is omitted, the link is assumed to be updatable. If required, time codes can be written on the arrow stem to show when the link is updated. For instance, it might be appropriate to build a link only on a specific user request; or to rebuild a link with a housekeeping run overnight to reflect structural changes as a result of the updates on a file.

Two or more links selecting items from a set only select the same item if the stems of the representative arrows share the same arrowhead. Thus, in Figure 3, the link from X is to the master item of set C, whereas from Y the link is to an item, Z, in that set.

DATA STRUCTURE

The notation developed in previous sections has been based on items in different sets. However, much of the complexity of data arises from links between items in the same set, i.e., the item structure, or between sets as a whole, i.e., the control structure.

Data architecture representation	VENN diagram
Set A ●●●●●●●●● **Set B** Zero or more items of Set A may be members of Set B	
Set A ●●●●●●●> **Set B** Every item of Set A is an item of Set B	
Set A <●●●●●●●●> **Set B** Every item of Set A is an item of Set B and vice-versa	
Set A ● Type ●●●●●●●> **Set B** ● Type=x Every member of Set A is either an item of Set B or an item of Set C ●●●> **Set C** ● Type=y	

XBL 7411-8308

Figure 2. Symbols for set inclusion relations.

Formalism

X ——→ Y. Item X always contains a link
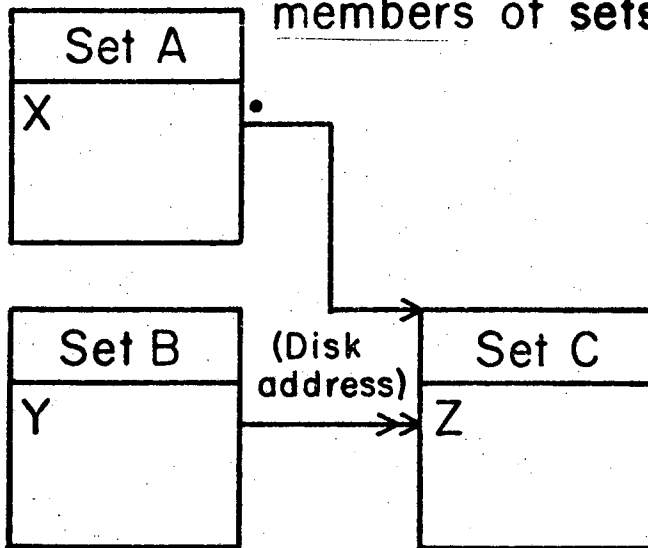   to the master item Y (e.g. queuehead)

X ---→ Y   Item X may contain a link as above

X ——→→ Y   Item X always contains a link to an
           item Y which is not a master item

X •——→ Y   Link XY is fixed when item Y is written

X $\xrightarrow{x}$ Y . Link XY is fixed at time x

Example: In which the items X,Y and Z are
           members of sets A,B and C

XBL7411-8306

Figure 3.   Symbols for linkage.

Sets can be organized in many ways -- as queues, chains, lists, stacks, tables, and serial files. We define a structured set as being one whose items each belong to a unique structure and whose items in total make up one or more structures lying completely within the set. The prime characteristic of a structure is that its items are ordered and associated in such a way that it is possible to access data items. Thus, when a structure is implemented in a computing system there must be a mechanism for :

- determining the first item, or head of the structure;

- recognizing the last item or structure tail;

- selecting the next item within the structure.

Thus, the concepts of structure and control are tightly interconnected, in that when defining the storage structure it is also relevant to define "from which data set or program is the description or instruction provided". We, therefore, developed the representation shown in Figure 4 to define structures using their interaction with a master set, via a 'control' linkage.

The double pronged control linkage indicates that each item of the master set controls a structure of the sample structure set. The two prongs of the control represent the ingress and egress attributes of the structure. The various values of these attributes that can be represented are shown in the Notation Summary Table.

Thus, it can be seen that the "Sample Structure Set" is a stack, in which items are added to, and removed from, the head. Structured set B is a table, in which each item of A contains a control for a structure of set B. The value of that control (and hence the order and number of items in the structure) remains fixed throughout the life of the master item. The control structure of the task control block vis-a-vis the data extent block chain may be null, i.e., the chain may be empty. The chain has items added to the head at time 0, and removed from anywhere within the chain at time C.

Not all linkages between sets are of the control type already discussed. A very important element of data structure is the linkage between items in different sets, or between different structures in the same set. This type of relationship and hence linkage is conveniently discussed using family relationships as in Figure 5.

The first diagram shows that the father may have an eldest son. The relationship is to this 'father's children' structure: and is represented by a single arrow. He may also have an eldest nephew, but the link would be in this case to an item in the set 'father's children' which was not under his (but under some other father's) control. This is shown by the double arrowhead, representing connection to another structure in the 'father's children' set.
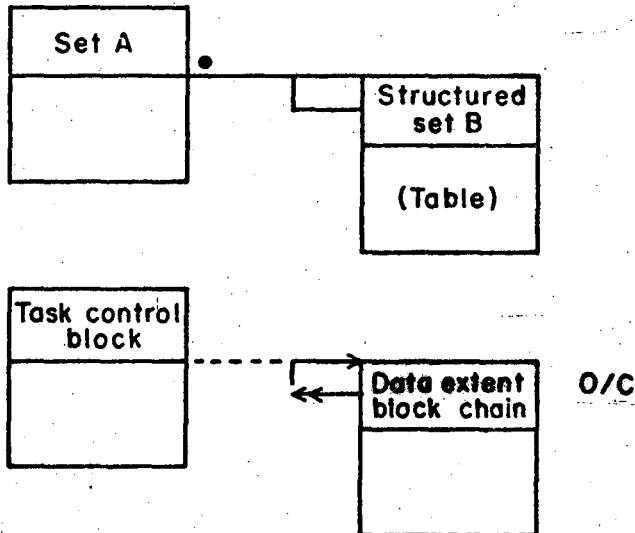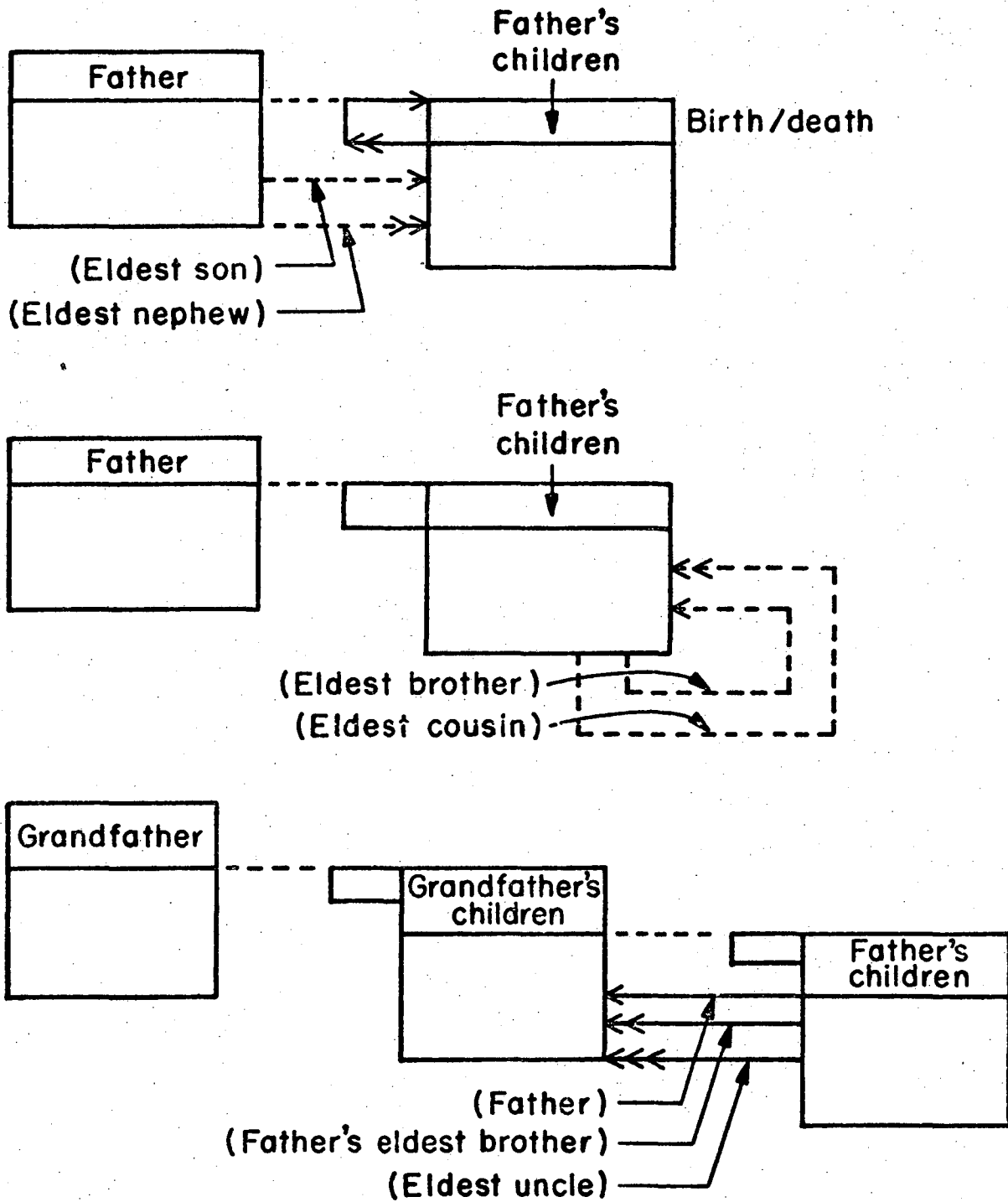
General

Master set	Control
	Ingress →
Stem	Sample
Egress ←	structure set

} A structured set
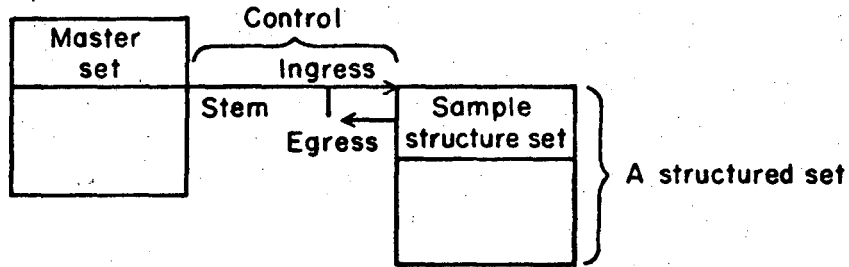
Notation summary table

Ingress notation	Egress notation
——→ Items added to head	←—— Items removed from head
——→‖ Items added to tail	⊦—— Items removed from tail
——→‖> Items added to head and tail	<⊦— Items removed from head and tail
——→> Items added anywhere	≪— Items removed from anywhere
—— Items not added	—— Items not removed

Examples

Set A	●
	Structured set B
	(Table)

| Task control block | |
| | Data extent block chain | O/C |

XBL7411-8304

Figure 4. Symbols for data structure representation.

Father's children

Father

Birth/death

(Eldest son)

(Eldest nephew)

Father's children

Father

(Eldest brother)

(Eldest cousin)

Grandfather

Grandfather's children

Father's children

(Father)

(Father's eldest brother)

(Eldest uncle)

XBL 7411-8309

Figure 5. Examples of linkage between and within sets

The middle diagram represents the linkage to items in the same set. Again, the single arrowhead is used to represent links to items in the same structure -- in this case the 'father's children' structure. The double arrowhead represents links to items in other structures of the 'father's children' set.

The last diagram in Figure 5 shows an extension of the linkage conventions to indicate 'another structure' within a master set. The use is clarified again by a geneological example. "Grandfather's children' is a master set of 'father's children'. Taking a given 'father's child': it contains a single arrowhead link to 'father', which is a master item. The link to 'father's eldest brother' has a double arrow-head because 'father's eldest brother' resides in the same structure as 'father', namely common 'grandfather's children'. However, 'eldest uncle' (which might be 'mother's brother') does not necessarily reside in the same structure as 'father'. This relation is represented by a triple arrowhead.

RELATION OF DATA AND PROGRAMS

So far we have been concerned with data relationships exclusively. While not viewing programs only as ancillaries of data, it is often useful when designing interrelated data structures or data architecture, to specify the programs which may access the data items or sets.

The main distinction, of course, between programs and data, is that a program has a flow of control. Illustrating this passage of control between programs in not the prime function of charts showing data architecture. When, however, the chart can be enhanced by showing passage of control, it is represented by arrowed doubled lines supported by comments which take the form of text in parentheses.

Figure 6 shows, as an example, a user program passing control to the EXCP supervisor program via the FLIH and the SLIH. It passes as a parameter the address of an IOB (Input/Output Block). The EXCP super-visor returns to the user by means of SVC exit.

CONCLUSION

The data structure charting methods described in this paper are part of a data architecture charting methodology. The methodology has been used extensively, as a

- design tool,

- teaching aid,

- documentation tool.

The scheme as a documentation tool has been particularly useful in analyzing systems developed elsewhere, for instance to extend or modify
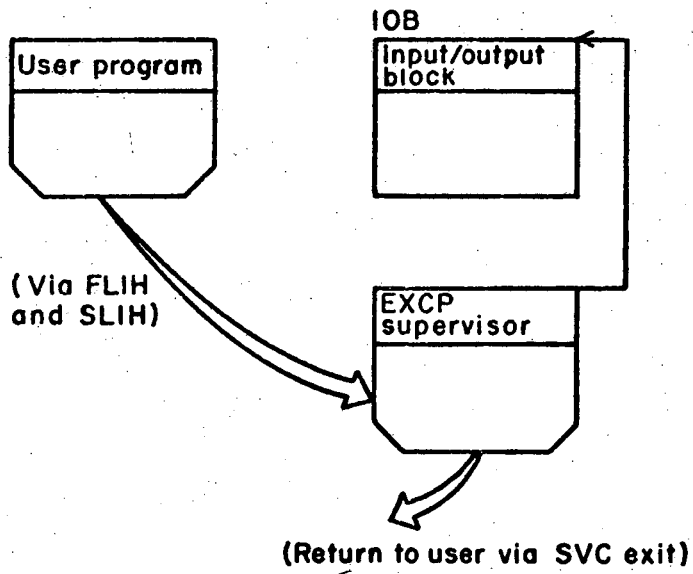
XBL 7411-8305

Figure 6. Representation of programs and data structures.

them. The scheme, as a method of introducing new staff to an existing
system or as an aid to explaining the design strategy to a nontechnical
manager, has proved itself many times over.

     However, the most urgent reason for adopting a structured method-
ology, such as this, is in its contribution to systems design. Designers
carry an awesome responsibility, and may soon be more formally accountable
for their products. Software designers should be aware that audits are
becoming more common; they should be worried by their lack of defense
against charges that software failures could have been avoided by the use
of better design methods.

ACKNOWLEDGEMENTS

     The authors would like to acknowledge that the ideas described here
have been developed by the Advanced Systems Design Group in CAP, over the
span of many projects.

REFERENCES

1.   Gill Ringland, "Design Tools for Data Handling Systems", LBL-3655,
     January 31, 1975.

2.   CAP, "The Data Architecture Charting Method Reference Manual",
     1975, available from either of the authors.

3.   CAP, "The Data Architecture Charting Method Tutorial Notes", 1975,
     available from either of the authors.

4.   Birkhoff and McLane, "A Survey of Modern Algebra", MacMillan, 1953.