# SOME COMMON ISSUES IN DATABASES FOR ENGINEERING DESIGN

by

W.J. Rasdorf, P.E., S.J. Fenves, and M. Locke

DRC-12-02-80

June 1980

Design Research Center
Carnegie-Mellon University
Pittsburgh, PA 15213

This work was partially supported by NSF Grant MCS-7822328.

6⇒0.0042
a28d
0 Rt-n-02-83

## Table of Contents

## List of Figures

## List of Tables

# 1. IHTEODUCTION

## 1.1 Purpose

The purpose of this report is to illustrate some of the concepts and research issues addressed in the research project, "Database Methods for Engineering Design".

Two specific examples will be used, one dealing with structural engineering design and one with chemical process design. In both examples, we will first introduce a design situation at a particular stage of the design process, describe the data representing the design at that stage, and define the functional integrity relations among the data. Subsequently, the design situation will be extended both "downstream" to subsequent design stages and "upstream" to earlier design stages, and the integrity relations among the data will be investigated.

The structural engineering design example will be further extended in order to discover issues which currently present the greatest problems in effectively implementing a comprehensive engineering design database. Many such issues surfaced from within the structural design example study and others became evident when consideration was given to incorporating the structural design example database into the entire building database [4].

## 1.2 Scope

A structural design example is presented in Chapter 2. In the example a structural design situation is introduced at a particular stage in the design process. This is followed by a description of the data representing the design at that stage, a definition of the functional integrity relations among the data, and a presentation of the tasks (application programs) which can operate on the data. The example is extended to discuss issues both prior and subsequent to the particular stage of design. Some of the major issues arising in the creation of an engineering database are then summarized. A chemical process design example is introduced in Chapter 3.

Chapter 4 introduces the CODASYL network model and describes the logical view of the entire frame operations system [7]. It presents and discusses the major areas (data items, integrity procedures, data manipulation language, data users) that comprise the system and the relationships between them. It explicitly describes the database and its contents within the context of the global operations system.

The model of the data to be stored in the database is also presented in Chapter 4« It includes the schema for the logical network data model and the data item type records for the physical data representation.

# 2. A STRUCTURAL DESIGN EXAMPLE

The following structural design example is introduced to provide a background to the reader in the concept of design in structural engineering. A subset of all of the possible constituents of a building--its frame--has been chosen as a stand-alone unit that is of sufficient size and complexity to represent the structure of a required database for design. The frame subset brings to light many integrity issues both within the subset and in its relationship to an entire building database. The example describes the elements of the physical structure, elements of the database, the application programs, the design process, and integrity issues which arise both in the database (data) and in the design process (application programs).

## 2.1 Design Example Assumptions and Constraints

The data structure included herein is one for an engineering "design" database. It has not been developed as a "comprehensive building" database. A comprehensive model would include considerations such as cost, construction schedules, maintenance, fire resistance, etc.. In addition, it would take into account the data needs of design disciplines. A comprehensive building database would also include alternative data structures for different possible structural systems. The database considered is thus a subset of a structural design database for steel building frames, but whose scope and interaction with other disciplines is sufficiently complex to allow for a meaningful study.

To further describe the scope of the database developed herein another important distinction must be made between a design and a comprehensive database. That difference lies in the level of detail of data needed by each. A comprehensive database would include dimensional information for detailing and fabricating steel members. Such specific dimensional information is not needed for design -- it is derived from the results of the design process, e.g., the designation of the standard sections selected.

## 2«2 The Physical Frame

The data structure of this project was developed for steel frames in two dimensional space. The steel frame may be braced by either rigid member connections or steel members used as diagonal bracing. No other bracing system such as multistory cores or shear walls is considered. The database was developed for the case in which the steel frame provides all structural support.

Figure 2-1 is included to familiarize the reader with the terminology used in the description of the frame. The frame is assumed to consist of three types of members: beams, columns, and bracing. Horizontal frame members are beams, vertical members are columns, and diagonal members are bracing. All members are assumed to be connected together in one of three ways: simple, rigid, or continuous connections. Simple connections are those which are completely flexible and are free to rotate in a manner similar to the action of a hinge. A rigid connection is one which does not permit relative rotation of the connected members. A continuous connection is one in which the members on each side of the connection are the same through the connection.

Some limiting topological assumptions have been made. In the future these limitations will be removed. Presently they are:

- Floors may not be displaced vertically. This results in all horizontal members being in a horizontal line at each floor level;

- Bays may not be multiple stories in height;

- Bracing members may intersect other members only at beam and column intersections.

Most of these limitations are caused because the current geometric representation utilizes only story height and bay- width to define geometry. Although not implemented in this study a more flexible and comprehensive geometric representation of joint coordinates would eliminate these shortcomings.

## 2,3 Initial Situation

At a particular stage in the design of a steel building, one of the frames comprising the building may be represented as shown in figure 2-1 . The representation of the frame consists of the following data types:

| Mnemonic | Description |
|---|---|
| COOED | Coordinates of joints defined by story height and bay width, |
| TOPO | Topology, i.e. connectivity of members and joints, |
| LOAD | Loads applied to frame (e.g., gravity, wind), |
| PROP | Section properties of members (e.g., area, moment of inertia), |
| DEFL | Deflections of joints, |
| FORCE | Force resultants in members, |
| SHAPE | Designation of standard structural components used for members, |
| CLEAR | Constraints on clearances for members, |
| LIMIT | Constraints on deflections for joints. |

The details of the data representation within each data type are not important issues for this presentation, except for some comparisons to be made in section 3» The "external" (user-level) representation for portions of the data is sketched in table 2-1 in as STRESS-like language [3]-

The general objective of this stage of design can be stated as follows:

```
D1:  Given:    COORD, TOPO, LOAD, CLEAR and LIMIT
     Obtain:   SHAPE (i.e., name of components chosen)
     Such that:DEFL consistent with LIMIT
               PROP consistent with FORCE
               Dimensions of SHAPE consistent with CLEAR
```

Since the frame is statically indeterminate, a direct solution of the design problem D1 is not possible, and the solution must be obtained by iteration. The following "unit operations" (application programs) may be employed:

D1.1 - INITIALIZE:
Given: COORD, TOPO and LOAD
Obtain: Initial values of PROP by some approximate procedure.

D1.2 - ANALYSIS:
Given: COORD, TOPO, LOAD and PROP
Obtain: DEFL, FORCE

D1.3 - SIZING:
Given: FORCE
Obtain: New values of PROP, not necessarily involving specific component selection

D1.4 - PROPORTIONING:
Given: FORCE, DEFL, LIMIT and CLEAR
Obtain: SHAPE

In typical iterative processes, there may be a few initial cycles of ANALYSIS-SIZING (treating PROP as continuous variables) until the system is reasonably stabilized, followed by cycles of ANALYSIS-PROPORTIONING to select actual components. The precedence relationship between the data types and the operations is shown graphically in figure 2-2 with rectangles representing the processes and ellipses representing data types. The relationship can also be represented by an incidence matrix. A, defined as

$$a_{ij} = \begin{bmatrix} +1 \\ -1 \\ 0 \end{bmatrix} \text{if data type } j \text{ is} \begin{bmatrix} \text{input to} \\ \text{output from} \\ \text{not used in} \end{bmatrix} \text{operation } i.$$

The matrix for the individual operations, as well as for the the design operation as a whole[1] is shown below.

[1]The rule for combining row entries in A is [6]: if data type output from an operation is only used as input to other operation(s) in group being combined, the resulting entry is 0; if it is also an output from the group, the entry is -1.

| | COORD | TOPO | LOAD | pROP | DEFL | FORCE | SHAPE | CLEAR | LIHIT |
|---|---|---|---|---|---|---|---|---|---|
| Initialize | 1 | 1 | 1 | -1 | | | | | |
| Analysis | 1 | 1 | 1 | 1 | -1 | -1 | | | |
| Sizing | | | | -1 | | 1 | | | |
| Proportioning | | | | -1 | 1 | 1 | -1 | 1 | 1 |
| DESIGN | 1 | 1 | T | | | | -1 | T | 1 |

## 2.4 Consistency and Integrity

If the entire design operation D1 were executed as a single step, no consistency issues would arise: the output, SHAPE, is consistent with input data types and the constraints. However, when the individual "unit operations" are performed in an iterative manner, each application of a unit operation, while enforcing consistency between the output data and the direct inputs, automatically invalidates consistency between the outputs and other data types. In the specific design example, the following situations occur:

1. after SIZING, consistency between PROP required and FORCE will be valid, however, consistency between PROP required and DEFL and between PROP required and PROP assumed is invalidated (i.e., the new PROPerties may not produce the same response);

2. after PROPORTIONING, consistency between SHAPE and all other data types is invalidated: i.e., the new SHAPE selected may not produce the same structural response, and may violate clearances; and

3. consistency between PROP provided by the SHAPES selected and PROP assumed is invalidated.

A modified relationship between the data and operations is shown in Figure 2-2. This figure explicitly distinguishes between:

1. assumed PROPERTIES which serve as input to ANALYSIS;

2. required PROPERTIES output by sizing; these may be directly input to ANALYSIS or passed on to PROPORTIONING; and

3. provided PROPERTIES of the SHAPES selected by PROPORTIONING.

In order to determine whether consistency has actually been violated, it is necessary to compare inputs and outputs and determine the magnitude of the changes induced; when these are judged sufficiently "small", the design process has converged.

## 2.5 "Downstream" Activities

After the design is judged satisfactory at the level stipulated in Section 2.4, additional activities take place.

In the stage usually referred to as DETAILING, the representation of the structural elements is extended by additional attributes, additional code and other consistency checks are applied, and additional objects (e.g., beam seats, connections) are defined and checked against their respective constraints.

Further downstream, during FABRICATION and ERECTION, the representation acquires other attributes (e.g., stage of approval, fabrication completion date, erection date, etc.).

Much further downstream, say, 20 years after completion, the representation may be used again when the structure is redesigned or retrofitted for a new usage.

## 2.6 "Upstream" Activities

The representation discussed in Section 2.3 arises fairly late in the overall DESIGN process of a building, and the data types identified as "inputs" are in fact outputs of preceding design stages. These preceding stages may be classified into three categories:

1. Spatial layout of the building based on its intended function. This step represents PRELIMINARY ARCHITECTURAL DESIGN which may be idealized for the purposes of the present discussion as the generation of COORD;

2. Selection of the structural scheme (e.g. steel vs. concrete; for steel, selection of "moment-resisting frame" vs. other possible framing types). This step represents PRELIMINARY STRUCTURAL DESIGN idealized as a first approximation of the generation of TOPO; and

3. Selection of initial structural properties. This step was

included in Section 2.3 as INITIALIZATION, but may in fact be a major separate process, possibly involving the separate approximate models for gravity and wind loads, and analyses with portions of the data assigned initial normative values. The net effect of this step is the generation of LOADs and preliminary values of PROPerties to initialize the DESIGN stage discussed in Section 2.3*

## 2«7 Iterations

The overall design process may include "global" iterations in addition to the "local" iteration described earlier. To cite a "worst-case" example, the dimensions (SHAPE) selected for the structural members may interfere so severely with headroom and clearance (CLEAR) requirements that the entire layout of the building must be changed, invalidating all results, since they are directly or indirectly dependent on COORD.

## 2.8 Partial Analyses and Redesigns

In the preceding discussion, it was implicitly assumed that any change or cycle of iteration would invoke a complete re-analysis or redesign of the system, i.e. the frame, in the present example. In actual practice, a designer may wish to "contain" the change to the portion of the system directly affected. For example, to investigate the effect of changing the depth of one beam in the frame, it may not be necessary or desirable to reanalyze the entire frame; it may be more appropriate to isolate a small portion for reanalysis (by assigning temporary, normative values on its boundary) or to use an alternate model.

At a broader scale, similar situations arise. The mere fact that one selects a single 2-D frame out of the complete 3-D building for analysis and proportioning is an example of partial analysis. Once a "typical" frame is designed and accepted, the other frames may be made identical or reanalyzed with local changes. The results of such partial designs may be accepted as the complete design, or may become inputs to a subsequent full 3-D analysis and redesign.

## 2.9 Summary of Implications

Although the details of the above example may be meaningful only to a structural engineer, it is hoped that the example will serve to illustrate many of the implications on the characteristics of a database for engineering design. These characteristics are briefly summarized below.

1. Information exists at various levels, corresponding roughly to the sequence of design stages, the information at each level representing a particular level of abstract description of the system.

2. Information grows as design progresses, i.e., as successive levels of abstraction descriptions are generated. The growth consists of the addition of new attributes to previously defined abstract entities; the generation of new entities "owned" by the higher-level ones; and the generation of new functional relationships among entities and their attributes.

3. Functional relations exist between information at different levels, as well as among data types within a given level.

4. The functional relations are either procedurally embedded into "unit operations" or can be represented as consistency requirements among data types. The question of whether and when to apply automatic procedures to insure consistency is highly problematical. See reference [5].

5. A number of mechanisms can be introduced to encode the status of the information in the database [2], e.g.; associate with the data various indicators or flags to distinguish:

   - "present" vs. "absent" data

   - "stage" or "permanence level" of data

   - "actual" or "normative" data (the latter usually an assumption or approximation, inserted to initiate an iteration or to provide a boundary value for a partial analysis).

   - "current" vs. "previous" for iterative data

   - "valid" vs. "void" with respect to the applicable constraints

   - etc.

Table 2-1: Functional Representation of Example Structure

```
STRUCTURE 'EXAMPLE1  'INITIAL TRIAL'
TYPE PLANE FRAME
JOINT COORDINATES
     1   X  0.0  Y  0.0  Z  0-0  SUPPORT
     5   X  0.0  Y 120.0 Z  0.0
     ...
MEMBER INCIDENCES
     1  FROM  1  TO  5
     2  FROM  5  TO 10
     ...
MEMBER PROPERTIES $ THESE ARE INITIAL VALUES
     1 THROUGH 7 PRISMATIC AX 100.  IZ 2000.
     ...
LOADING 1 'WIND'
JOINT LOADS
     2  FORCE  X  10.0
     5  FORCE  X  12.0
     ...
LOADING 2 'GRAVITY1
MEMBER LOADS UNIFORM FORCE Y
     15  ¥ -2.5
     30  ¥ -3.0
     ...
LIST FORCES ALL, DEFLECTIONS X 2,5,...
SOLVE
```

Figure 2-1: Topological and Geometric Frame Definition

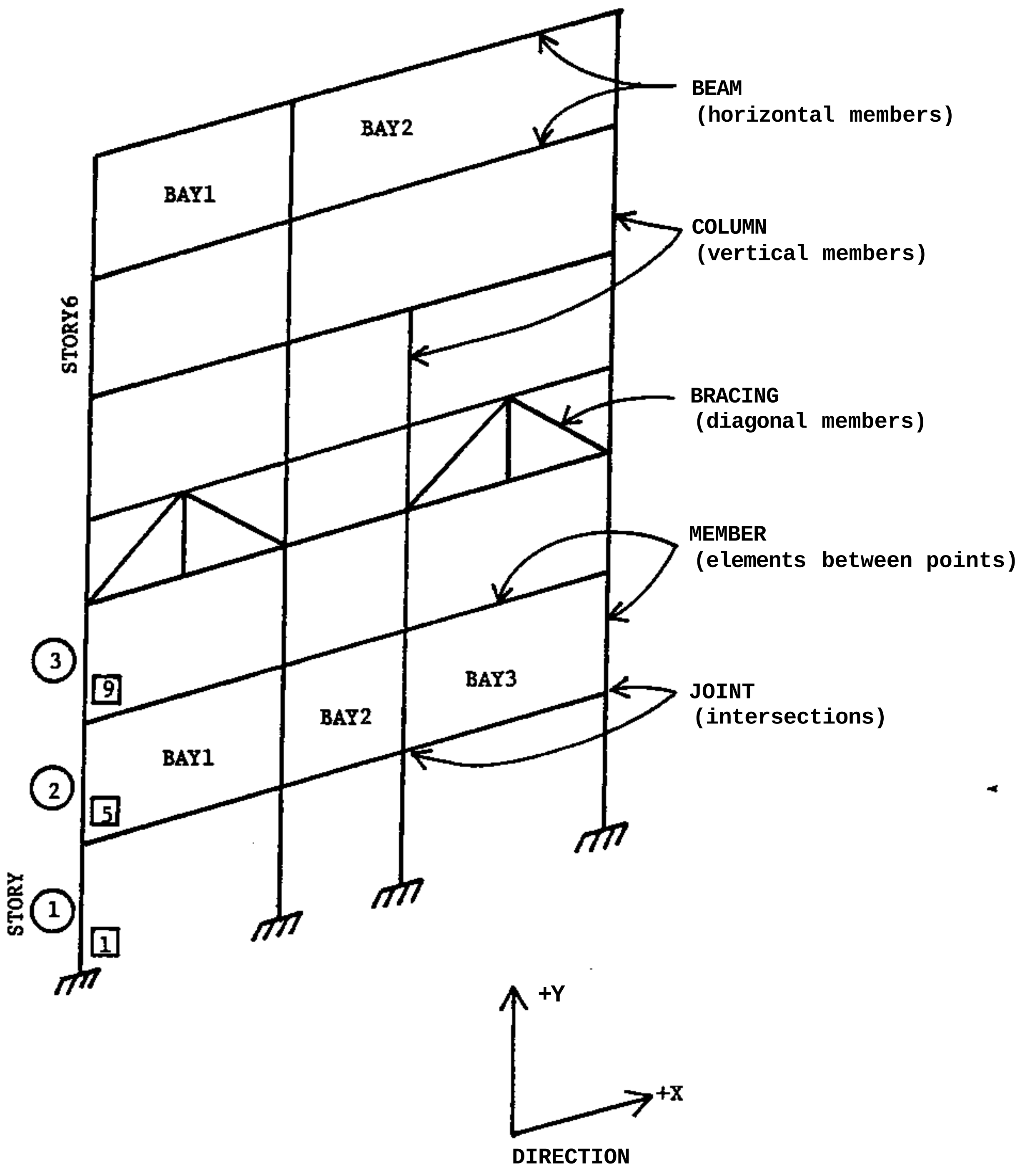
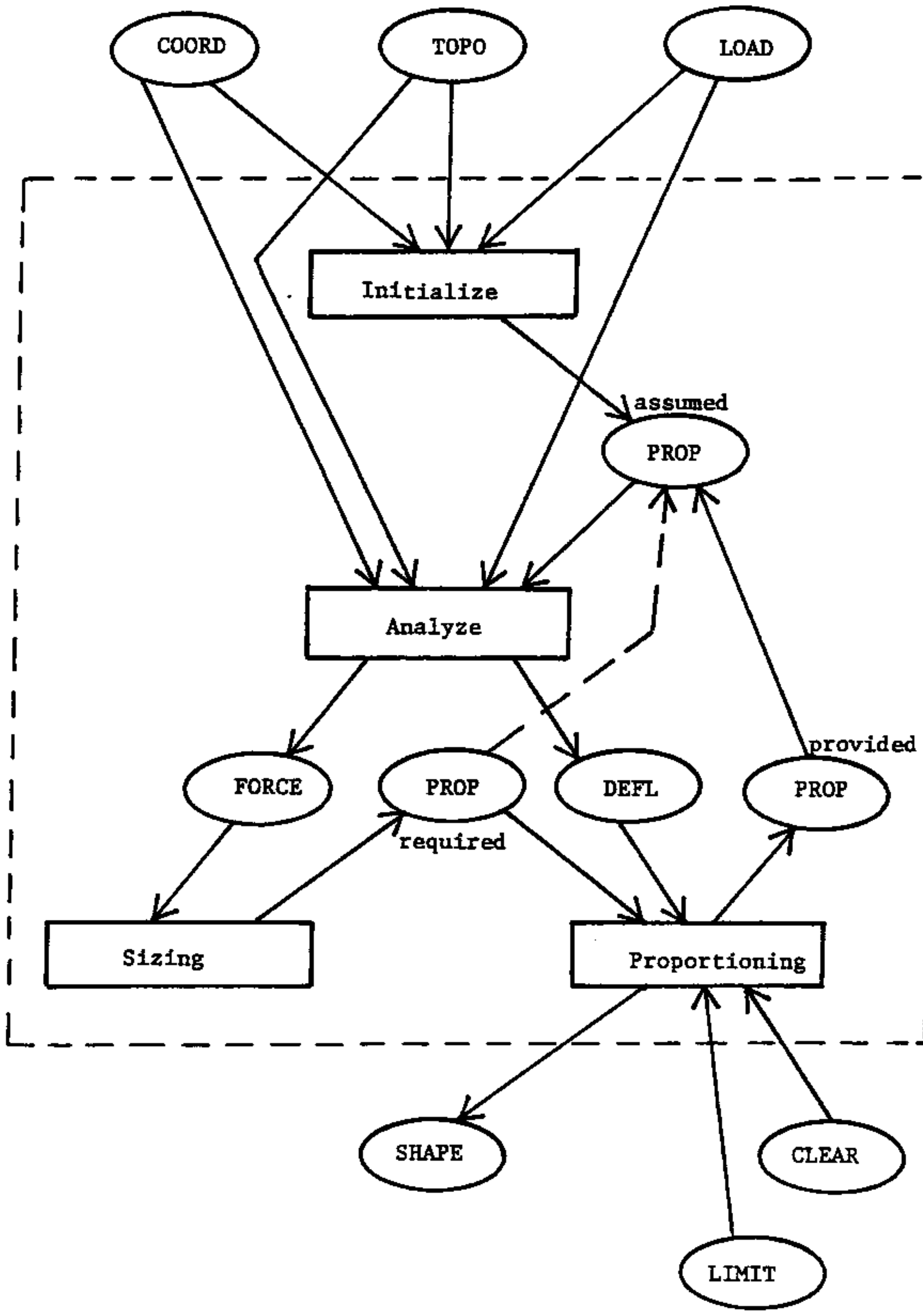


Figure 2-2: DESIGN Relationships: Process - Data



# 3- A PROCESS DESIGN EXAMPLE

## 3.1 Initial Situation

At a particular stage in the design of a process plant, one of the subsystems of the plant may be represented by the flowsheet shown in Figure 3-1 [Westerberg et.al, undated]. The representation of the flowsheet consists of the following data types (called variable packets in [ASCEND-2 An Advanced System for Chemical Engineering Design, 1980]):

| Mnemonic | Description |
|---|---|
| Si | Stream variables (flow rate, temperature, pressure) |
| PMPi | Pump variables (pressure drop, efficiency, etc.) |
| PlPi | Pipe variables (diameter, length, Reynolds No., etc.) |
| VISCi | Viscosity variable |
| EVi | Evaporator variables (no. of tubes, tube radius, etc.) |
| SPFRi | Splitter variable (split fraction) |

The "external" (user-level) representation of the flowsheet is shown in Table 3-1. The VP (Variable Packet) and PT (pointer) statements associate the stream variables to physical property variable packets AMMONia and WATER. The FS (flowsheet) statements describe what types of units (eg. pump, mixer, evaporator) are to be modeled. Finally, the V (variable) statements define the topology of the flowsheet, i.e., specify the inlet and outlet streams and internal variable packets of the preceding FS statement.

The objective of this step is to solve the steady state equations describing the flowsheet. To do so either a simulation or a design calculation can be performed.

In a simulation or rating calculation all system inputs and design parameters for the units are specified. Variables such as pipe length and diameter are specified. Outlet streams and internal streams of the flowsheet are calculated.

In terms of the flowsheet of Figure 3«1f a simulation calculation is related to the operation:

D.2.1 - ANALYSIS:

| | |
|---|---|
| Given: | Flowsheet, Equipment described, input streams S1 and S5 |
| Obtain: | Internal and Outlet stream variable values |
| Such that: | Steady state mass and energy balance equations are satisfied. |

In a design calculation, outputs are specified and inputs or unit parameters are calculated based on these output specifications. It differs from a simulation calculation in that different variables are fixed and calculated.

The design objective for Figure 3-1 may be:

D.2.2 - DESIGN:

| | |
|---|---|
| Given: | Flow sheet, Inlet stream S1, Outlet stream S8 |
| Obtain: | Description of equipment (pumps, mixer, etc.) input streams |
| Such that: | Outlet stream S8 has requisite properties and material and energy balance equations are satisfied. |

In the ASCEND-2 program, the different calculations are specified by designating the appropriate variables as being either "fixed" or "calculated".

Until the overall design objective is accomplished, repeated simulations may be performed in an iterative manner. These simulations involve the same issues of consistency and integrity as discussed in Section 2.4« Specifically, the change of any equipment or stream parameters between successive simulations invalidates consistency with respect to steady-state equilibrium established by the previous iteration, and it is necessary to compare the results of the two simulations to determine the magnitude of the changes induced.

In addition to consistency of variable values, consistency in calculation type must also be maintained. Variables which are "fixed" in a simulation may be "calculated" in a design calculation. This possibility necessitates checking consistency of "fixed" and "calculated" flags.

## 3.2 "Downstream" Activities

After the design is judged satisfactory at the level of steady-state simulation, additional activities take place.

Still in the stage of process design, a dynamic simulation may be performed, so that the performance of the system may be evaluated in greater detail under start-up, shut-down, shock-loading and other dynamic conditions.

In the stage called PID (Piping and Instrumentation Design), the representation of the subsystem is extended by additional attributes. The entity "Pump", for example, acquires attributes such as power consumption, cost, control system description, etc.. Again, as in the structural example, information grows by additional attributes (power, cost) as well as by additional entities which may be complex subsystems themselves (control system).

Further downstream, the representation acquires still additional attributes (e.g., space, weight, location, manufacturer's identification, etc. of the pump), until finally delivery dates, installation, pilot operation, etc. data are associated with the entities.

Much further downstream, say 10 years after initial operation, the representation may be used again when the plant is redesigned or modified for, say, energy conservation, or even for an entirely new process.

## 3«3 "Upstream" Activities, Iterations and Partial Analyses

The representation of the subsystem discussed in Section 2.3 arise3 fairly late in the overall design of a chemical process plant. The design stages preceding the stage discussed may be classified as follows:

1. overall synthesis of the plant;
2. synthesis of the subsystem; and
3* initialization of the subsystem for simulation purposes.

The ASCEND-2 program permits a number of ways to accomplish the last step above. For example, one unit, such as the evaporator, may be isolated and a partial simulation performed by assigning temporary, normative initial values to the stream variables S4 and S6. Once this simulation is deemed satisfactory, its results may in turn become the initial values for the simulation of the entire subsystem.

In a similar fashion, when the entire plant is being simulated for system integration, the "inputs" S1 and S5 and "outputs" S8 and S11 of the subsystem will themselves become "internal variables" of the overall plant model. Conversely, the selection of the subsystem for separate simulation is, in itself, an example of partial synthesis.

## 3»4 Application of Constraints and Consistency Checks

It should be clear that the process design example possesses exactly the same characteristics as the structural example discussed in Section 2.7, and poses the same implications for a design database.

It appears clear that the mechanisms sketched in Section 2.7 can be quite directly applied to the example of the process plant. As an example, the variables needed for the simulation proper could be augmented by:

- additional attributes, e.g. "cost",

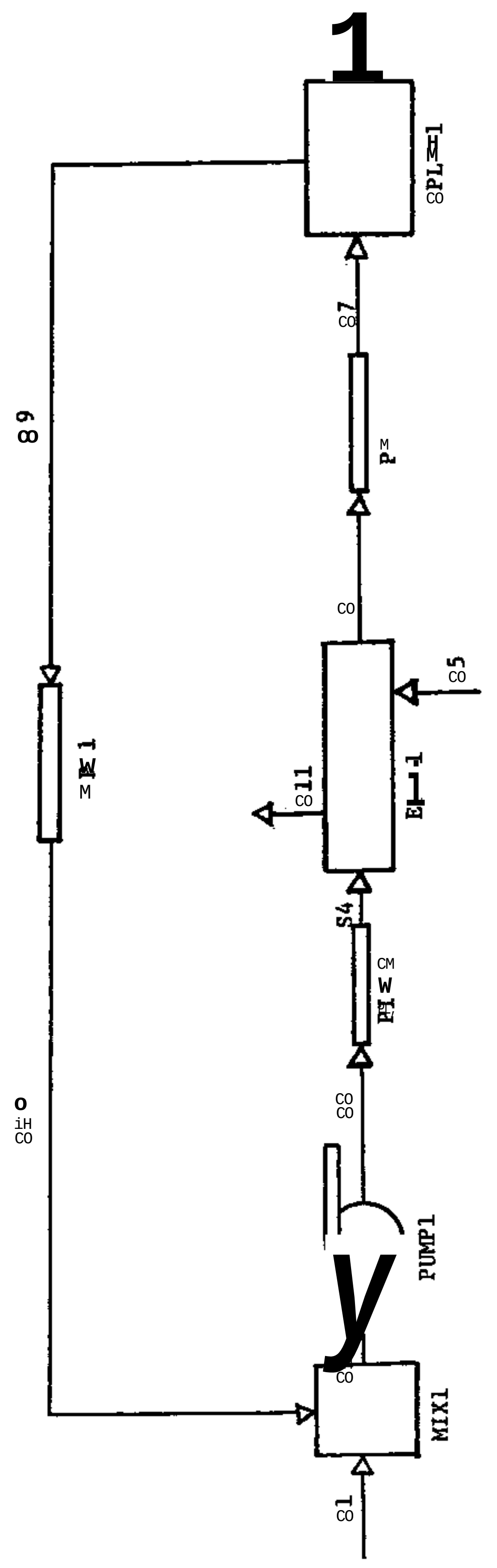
- constraints, as, for example, upper and lower limits on pressures, and,

- status indicators to: designate design state; distinguish between normative data inserted for initialization vs. those derived from the simulation; distinguish whether "valid" or "void" with respect to constraints, etc..

Table 3-1: Functional Flowsheet Representation

```
C*   STREAMS OF TYPE 1 ARE AMMONIA
VP   STRT1   AMMON
PT   S1      STRT1
PT   S2      STRT1
PT   S3      STRT1
PT   S4      STRT1
PT   S6      STRT1
PT   S7      STRT1
PT   S8      STRT1
PT   S9      STRT1
PT   S10     STRT1
C*   STREAMS OF TYPE 2 ARE WATER
VP   STRT2   WATER
PT   S5      STRT2
PT   S11     STRT2
C* START OF ACTUAL FLOWSHEET
FS   MIX1    MIXER
V    S1      S10     S2
G    MIXR
E    MIX
FS   PUMP1   PUMP
V    S2      S3      PMP1
G    PUMP
E    PMP1
FS   PIPE2   PIPE
V    S3      S4      PIP2    VISC2
G    PIPE    VISC
E    PYP2    VIS2
C* USE DEFAULT STATEMENT TO FILL IN EQN PACKS OF THIS UNIT
DF   DFLT1   EVAP
E    EVP1
G    EVP
FS   EVAP1           DFLT1
V    S4      S6      S5      S11     EV1
FS   PIPE3   PIPE
V    S6      S7      PIP3    VISC3
G    PIPE    VISC
E    PYP3    VIS3
FS   SPLIT1  SPLIT
V    S2      S8      S9      SPFR
G    SPLITR
E    SPMBAL
FS   PIPE1   PIPE
V    S9      S10     PIP1    VISC1
G    PIPE    VISC
E    PYP1    VIS1
C* NOTE THAT THE ORDER THAT THE UNITS ARE INPUT
C* IS ARBITRARY IF THE INITIALIZATION ORDER IS
C* LATER SPECIFIED.  IF THE INITIALIZATION ORDER
C* IS NOT SPECIFIED, THEN THE UNITS ARE INITIALIZED
C* IN THE ORDER THAT THEY ARE READ IN.
EN
```

Figure 3.1: Functional Flowsheet Representation



# 4. THE DATA MODEL

As an illustrative vehicle, a CODASYL network database model was developed for the structural design example of Chapter 2. The logical network structure was developed and the record types defining the data structure were coded in Pascal. This chapter details the data model. The model is based on the physical structure of the frame discussed in Chapter 2 and the conceptual structure of the operations system presented below.

## 4*1 The Operations System

Figure 2-2 illustrates how the data items, design processes, and checking procedures are conceptually related in the "design" system. Figure 4-1 illustrates an alternative representation of figure 2-2 showing how they are related in the "operations" system, a system in which the information needs of the user are anticipated at the time the system is designed and application programs are written to handle those needs in an efficient manner [7]. The figure is divided into three major conceptual areas consisting of the Database, the Data Manipulation Language, and the Data Users.

The database consists of a collection of interrelated data items. These data items are stored together so that they are independent of the data users. They have a controlled degree of redundancy[2] and a common approach is used in adding new data and in modifying and retrieving existing data [7]. In addition the database contains some of the integrity procedures used to check the data.

The purpose of integrity procedures is to insure that incorrect

[2] Redundancy can never be totally eliminated. The system is therefore designed in such a way that it is controlled and minimized.

data items are not stored in the database. Some integrity procedures are in themselves data item constraints that logically fit in the database. Other integrity procedures, however, such as one that checks data items against a design code, are more general and may apply to a class of many buildings. These could be removed from the database and introduced as a group of procedures represented as a separate application program. The division between the two kinds of integrity procedures could be as simple as dividing between general building procedures and specific building procedures. This issue is one which requires further study.

INPUT, MODIFY, and OUTPUT are the three primary categories of operations on the data items of the database. Examples of commands within these categories are shown in figure 4-1. These and similar commands comprise the data manipulation language. Externally the ML is a list of allowable co""""»nds that initiate predefined tasks or operations on the database and, internally the DML is a collection of procedures which actually perform the tasks or operations. Thus the DML is a mechanism of communication between the database and the data users.

The primary database users are individuals and application programs who wish to enter, modify, and extract data. They do so by utilizing the data manipulation language discussed in the previous section. The DML should be designed to support both types of uses.

## 4*2 Logical Data Representation

Figure 4-2 represents the network data model for the data items shown in figure 2-2. The additional data items in figure 4-2 not previously present are discussed in section 4«3« The lines between the data items represent links or relations between them. The direction of the link indicates a relationship of one to many from tail to head. For example, each frame contains many members but each member is in only one frame. The schematic result is a directed graph.

In the network model many to many relationships are not permitted. To represent a many to many relationship between two data item types a "dummy" type is introduced between them. For every occurrence of a relationship between the original data item types there exists a data item instance in the dummy record linking them together. In figure 4-2 the data item relationships connected by dummy record types are bays to members and members to joints.

The links of figure 4-2 were chosen to establish accesses between data item types that were deemed most useful in structural design. To establish additional relations would incur an additional cost for the necessary pointers needed to physically establish the links. Additional relations need not be defined, however, because in most cases the same result can be obtained by traversing multiple existing links. If a need arose for such accesses between indirectly linked data item types, procedures could be written to perform the accesses within the framework of the existing network model.

## 4*3 Data Representation

The following sections discuss aspects of the structure of the data base. Included are all the data items of the frame shown in figures 2-2 and 4-2. The physical representation manifests itself in the form of Pascal coding. Records and other type declarations as well as important variables were coded. These are presented in the Appendix. The discussions which follow point out the meaning of the data items, their attributes, and the relationships between them. Table 4-1 summarizes the form of physical implementation of each of the data types.

### 4.3.1 TOPO

Topology describes the connectivity of the physical components of a structure, i.e. to which other elements a given element is connected [i]. For the frame the topology provides the primary structure of the data base to which all of the other information is attached (COORD, SHAPE, PROP, LOAD, FORCE, DEFL).

The topology can be viewed at three levels. The schematic connectivity is illustrated in figure 2-1 in which all of the frame components are shown. The connectivity of the network model or conceptual schema is shown in figure 4-2. Finally, the connectivity in the implementation is represented by the coded records in the Appendix.

The topology of frame, story, bay, member, and joint entities and all of their connecting relations forms the structural base to which all other data entities can be attached. The implementation establishes this base by its records and the pointers which link them together. Thus TOPO is represented by records. Each of the components of the physical structure (frame, story, bay, member, joint) is a record. External access to the topology, and consequently to most of the data structure, is provided by a system pointer to the frame (headframe). Internal access to items within the data structure is gained from the frame by following the pointers to succeeding records until the desired one is reached.

**4*3.2 COORD**

Geometry describes the physical dimensions and location in space of each component of a structure [i]. In this database geometry is represented in terms of COORD.

To eliminate joint coordinate redundancy it was determined that one need only store the height of each story and the width of each bay (based on assumptions stated previously). A procedure could then be developed to provide all necessary geometric information based on these values. Thus COORD is represented by a bay width field in the bay record and a story height field in the story record. Geometric information is accessed in the data structure through the topology.

### 4.3-3 SHAPE

The end result of the design process is the selection of a shape for each member in the frame. The physical manifestation of this result in the database is obtained by setting a pointer from each member to a SHAPE record. There is a single shape record for each member. The shape record is an intermediary which subsequently points into a table (containing all possible shapes) to a specific shape. In this manner each table of shapes is independent of the database. The database is structured so that the shape records and the entries in the shape tables are accessed externally of the frame by means of system pointers to them (headwideflange, headchannel, headangle, headtee, headshape). Member records, however (part of the topology), must be accessed through the frame record.

### 4.3.4 PROP

There is an important distinction between PROP and SHAPE. While the tables of SHAPE used by the Proportioning application program contain information about the properties for each fabricated shape (torsional constant, dimensions, etc.) the PROP records used by Analyze and Proportioning have fields for only the design performance properties of each member (area, section modulus, etc.). PROP is a record related to and accessed from the member records of the frame.

### 4.3.5 LOAD

It is assumed that all wind and gravity loads applied to the frame may be introduced in the data structure as loads on members. Each member may have multiple uniform and concentrated loads. In addition loads may be associated with members for multiple loading conditions.

LOAD is represented as records. There exist linked lists of many loads and load conditions for each member. Each load is an intermediary pointing to a single specific (uniform, concentrated, or triangular as determined by loadtype) load. Each load record is accessed from the member records of the frame through a loadcondition record. That is, from member there is a pointer to a loadcondition record that in turn points to each of live, dead, and wind load records. Within the load records there are pointers to the next live, dead, and wind loads in the list so that the user has direct access to all loads or to all loads of a given kind, for every loading condition, for every member.

### 4.3.6 FORCE

Force resultants are the forces stored in the database. For members in a two dimensional frame there is an axial force, a moment at each end, and a shear at each end. These are the stored applied forces.

FORCE is represented as records. There are a minimum of three force records for every forcecondition record, each of which corresponds to live, dead, and wind forces. Each force condition corresponds to the loading condition which induced the given forces. FORCE is accessed from the member records through a forcecondition record.

### 4.3.7 DEFL

The data structure stores joint deflections. DEFL is a numerical value which is stored in a field of the joint records. Vertical (y) and horizontal (x) deflections and rotations relative to the undeflected frame are stored.

Table 4-1: Data Item Implementation

| DATA | IMPLEMENTATION |
|---|---|
| TOPO | Pointer fields connecting frame, story, bay, member, and joint records accessed by pointer from system. |
| COORD | Field in bay record.<br>Field in story record. |
| SHAPE | Record accessed by pointer from member and system. |
| PROP | Record accessed by pointer from member. |
| LOAD | Record accessed by pointer from member (through loadcondition). |
| FORCE | Record accessed by pointer from member (through forcecondition). |
| DEFL | Fields in joint and member records. |
| CLEAR | Integrity procedure. |
| LIMIT | Integrity procedure. |

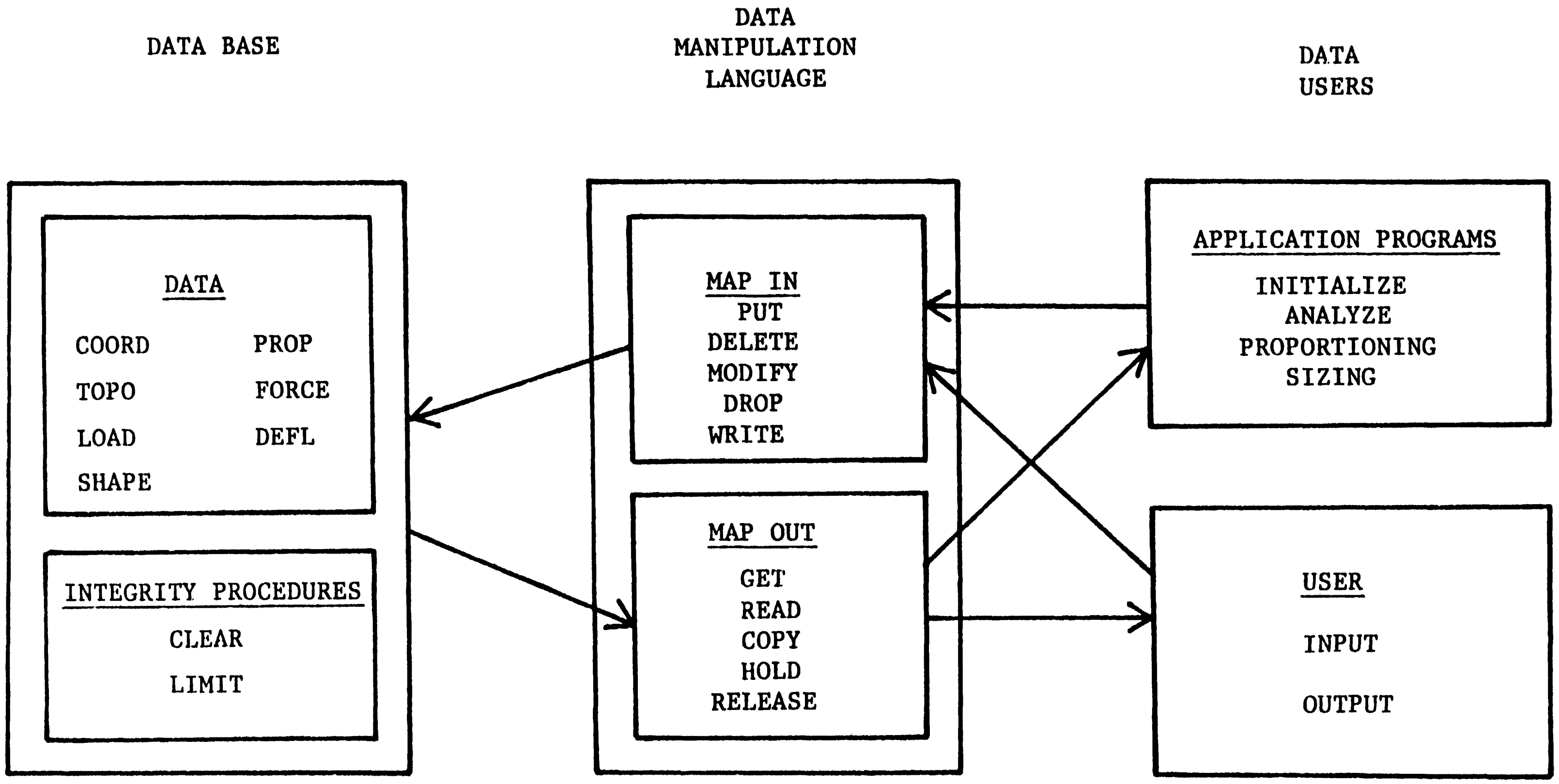


Figure 4-1: Operations System

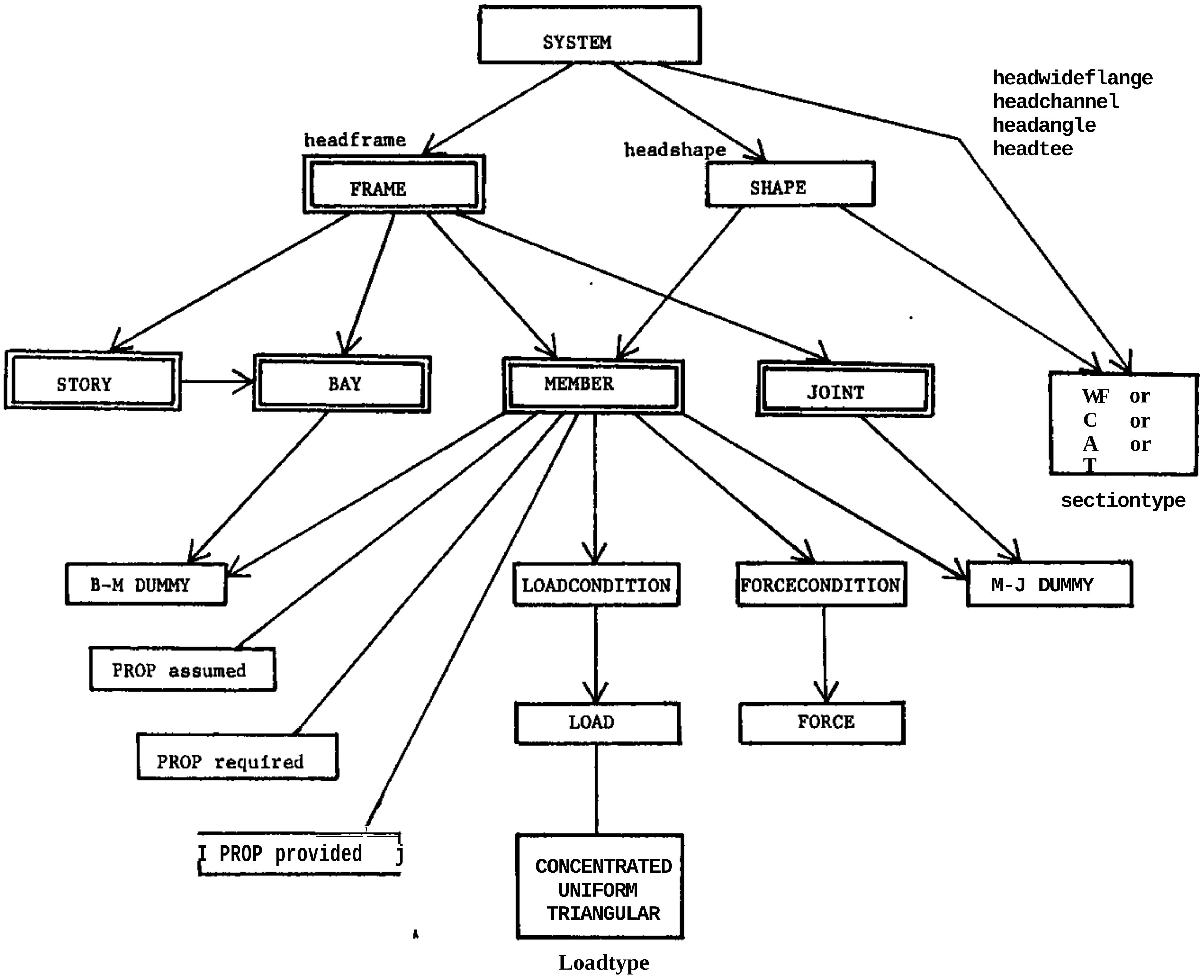


Figure [illegible]: Network Structural Schema

## REFERENCES

[i] Baer, A., Eastman, C, Henrion, M.
Geometric Modelling: A Survey.
~~Proceedings CAD79 IPC Buisneas Press~~ 11(5):253-269, September, 1979.

[2] Eastman, C. M., and Fenves, S. J.
~~Design Representation and Consistency Maintenance Needs*~~
Research Report 75, Institute of Physical Planning, Carnegie-Mellon University, May, 1978.

[3] Fenves, S.J., Logcher, R.D., Mauch, S.P. and Reinschmidt, K.F.
STRESS - .A User's Manual.
M.I.T. Press, Cambridge, MA, 1964.

[4] Keyvanfar, Faramarz.
Design of Multi-Story Combined Wall-Frame Buildings.
Master's thesis, Carnegie-Mellon University, Pittsburgh, Pennsylvania, August, 1980.

[5] Lafue, G. M. E.
~~Integrating Language a~~nd Database for CAD Applications.
Computer Aided Design 11(3):127-130, May, 1979-

[6] ~~Langefors, B.~~
Theoretical Analysis of Information Systems.
Studentlitteratur, Sweden, 1970.

[7] ~~Martin, Jam~~es.
Principles gf Data-Base Management.
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1976.

# APPENDIX

This appendix contains the coded physical representation of the data item types of the database. It represents the frame entities, their attributes, and their relationships to each other. In particular it includes:

- standard and pointer type declarations,
- record type declarations enumerating the fields of the data items,
- variable declarations of variables important to the data structure, and
- procedures for finding information in the data base.

TYPE

```
        SECTZONTYPE       «i (WF, C, A, T)?
        LOADTYPE          i« CCO, UN, TR)f
        MEMBERTYPE        i• (beam, column, bracing)?
        CONNECTZONTYPE    «i dimple* rigid, continuous)?
        NAME              i• ARRAY C1..10) OF char?

        {pointer typas)

        FRAMEPTR          •* •frame*
        STORYPIR          •: *story;
        BAYPTR            i' ¯bay?
        MEMBERPTR         i: *member;
        JOZNTPTR          <• ¯joint»

        LOADCONDITIONPTR  «• 'loadconditlon?
        LOAOPTR           *» 'load?
        CONCENTRATEDPTR   i• 'concentrated?
        UNZFORMPTR        «i •uniform?
        TRZANGULARPTR     *i 'triangular?

        FORCECONDZTZONPTR •• •forceeondition?
        FORCEPTR          i> "forcet

        SHAPEPTR          ii *shapa|
        WIDEFLANGEPTR     ii 'wldafianga?
        CHANNELPTR        ai *ehannel?
        ANGLEPTR          «i¯anglai
        TEEPTR            »i "tea?

        PR0PA3SUMEDPTR    «i •propassumedj
        PROPREQUZREDPTR   •i *propreguired?
        PROPPROVZOEDPTR   «' *propprovided?
```

```
FRAME • RECORD

        frameid    I  INTEGER?

        {pointer  variables)

        nextframe  ⁞  FRAMEPTRf
        ttorys     ⁞  STORYPTRl
        bays       ⁞  BAYPTRf
        members    ⁞  MEMBERPTRf
        joints     ⁞  JOINTPTRf
        END)

{nextframe  i  ntxt In framt 1inletd list
 storys     t  linked list of storyt
 bays       i  linked list of bays
 ntnbert    I  llnktd list of members
 joints     i  linked list of joints)
```

```
STORY • RECORD

        Itoryid      | INTEGER)
        storyheight  i REAM

        {pointer variables)

        nextstory    t STORYPTRf
        ttorybays    i BAYPTRf
        END)

istoryheight t centerline to centerline grid height
 nextttory   i next story in linked list of storys
 storybays   t linked list of bays in this story)
```

```
BAY • RECORD

          bayid            I INTEGER;
          bayvidth         I REAL>

          {pointer variables)

          nextbay          t BAYPTRf
          bayitory         t STORYPTRi
          nextbaythistory  i BAYPTRJ
          baymemben        I MEMBERPTRf
          END;

<baywidtn          » centerline to eenterllne grid width
 nextbay           I next bay in linked lilt of bays
 baystory          t story this bay is in
 nextbaythistory   t next bay in linked list of bayi for given story
 baymembers        t linked list of members this bay)
```

```
MEMBER = RECORD

        memberid              : INTEGER;
        memberdeflection      : REAL;
        deflectionlocation    : REAL;

        memberkind            : SECTIONTYPE;
        function              : MEMBERTYPE;
        endjoint1connection   : CONNECTIONTYPE;
        endjoint2connection   : CONNECTIONTYPE;

        {pointer variables}

        nextmember            : MEMBERPTR;
        memberbay1            : BAYPTR;
        memberbay2            : BAYPTR;
        nextmemberbay1        : MEMBERPTR;
        nextmemberbay2        : MEMBERPTR;
        endjoint1             : JOINTPTR;
        endjoint2             : JOINTPTR;
        nextmemberendjoint1   : MEMBERPTR;
        nextmemberendjoint2   : MEMBERPTR;
        nextmemberthiskind    : MEMBERPTR;
        assumedproperties     : PROPASSUMEDPTR;
        requiredproperties    : PROPREQUIREDPTR;
        providedproperties    : PROPPROVIDEDPTR;
        appliedloads          : LOADCONDITIONPTR;
        appliedforces         : FORCECONDITIONPTR;
        sectionchosen         : SHAPEPTR;
        END;

{memberdeflection      : maximum deflection of member measured from undeflected
                         positive in upward and rightward directions
 deflectionlocation    : distance of member deflection from joint1 in feet
 memberkind            : WF, C, A, T
 function              : beam, column, bracing
 endjoint*connection   : simple, rigid, continuous
 nextmember            : next member in linked list of all members
 memberbay1            : one bay this member is in (bay1)
 memberbay2            : other bay this member is in (bay2)
 nextmemberbay1        : next member (any one) in bay1
 nextmemberbay2        : next member (any one) in bay2
 endjoint1             : beams : left; columns : lower; bracing : left;
 endjoint2             : beams : right; columns : upper; bracing : right;
 nextmemberendjoint1   : next (any one) member at endjoint1
 nextmemberindjoint2   : next (any one) member at endjoint2
 nextmemberthiskind    : linked list of members made of given section
 assumedproperties     : assumed member properties
 requiredproperties    : required member properties
 providedproperties    : provided member properties
 appliedloads          : the externally applied loads on this member
 appliedforces         : applied forces on this member due to applied loads
 sectionchosen         : section this member is chosen to be}
```

```
JOINT • RECORD

         jointid          |  INTEGER?
         jolntdefltctlon t  ARRAY C1..2] of REALf

         {pointer vari«blts>

         nextjoint        t  JOINTPTRj
         jointnembers     i  MEMBERPTRJ
         END}

<jointdeflection  t  amount of deflection of the joint
 xdefieetlon      i  horizontal joint deflection (positive rightward)
 ydefiectlon      t  vertical joint deflection (positive upward)
 nextjolnt        I  next joint in llnlced list of joints
 jolntmembcrs     I  linked list of member! attached to this joint)
```

```
        PROPASSUMED • RECORD

                area                 i1 REAL!
                momentofInertlax     i1 REAL!
                sectionmodulusx      i1 REAL!
                momentofinertiay     I1 REAL!
                sectionfodulusy      i1 REAL)
                torslonalconstant    i1 PEAL!
                plastlemodulusx      iI REAM
                plattlcmodulusy      i1 REALf

       (this rteord contains th« REQUIRED values for lection properties
        for the member obtained from the DESIGN process* i.e. from either
        the application programs INITIALIZE or ANALYZE.
        This record contains only DESIGN values—no geometry.
        Geometric properties come from the SHAPE records for
        specific chosen shapes that are PROVIDED for the given member.)

****************************************

        PROPREQUIRED • RECORD

                (similar to propertyassumed)

                END)

****************************************

        PROPPROVIDED • RECORD

                (similar to propertyassumed)

                ENDf
```

```
LOADCONDITIQN • RECORD

        eonditionnumbtr   i REAM

        {pointer variables)

        nextloadcondltion j LOADCONDITIONPTRJ
        applledllveloads  t LOADPTRi
        applieddeadloads  i LOADPTRf
        appl1edwlndloads  t LOADPTRj

{eonditionnumber   i a number Indicating which load condition
 nextloadcondition i the next load condition
 applied»loadt     i the externally applied • loads on this member)
```

---

```
LOAD • RECORD

        {pointer variables)

        nextloadthlskind t LOADPTRf

        {variant)

        kind     I LOADTYPEl
        CASE loadtype OF
              CO I (cone I CONCENTRATEDPTR);
              UN t (unit I UNXFORMPTR)!
              TR I (trla I TRIAN6ULARPTR))
        END|

<next*load t next • load in linked list of loads
 kind      i loadtype on the member
 CASE      t record containing specific details about this loadtype)
```

```
      CONCENTRATED « RECORD

              magnitude      i REAL*
              direction      s REAL?
              distfromjolntl s REAL;
              END)

     (magnitude       t  amount of applied load in kips
      direction       i  measured from downward relative to frame
                          CCW i positive)   cw t negative)
      distfromjointl t  distance of load application from jointl in feet)
```

****************************************************************

```
      UNIFORM a RECORD

              magnitude        S REAL)
              direction        I REAL)
              startfromjointl i REAL)
              endfromjolntl   I REAL)
              END)

     (magnitude        i  amount of applied load in Kips per foot
      direction        t  measured from downward relative to frame
                           CCW i positive)  CW i negative)
      startfromjointl s  starting distance of applied load from jointl in feet
      endfromjolntl    t  ending distance of applied load from jointl in feet)
```

****************************************************************

```
      TRIANGULAR « RECORD

              maxmagnltude   t REAL)
              direction      t REAL)
              maxfromjointl t REAL)
              mlnfromjointl t REAL)
              END)

     (maxmagnltude  t  maximum amount of applied load in kips per foot
      direction     t  measured from downward relative to frame
                        CCW t positive)   cw i negative)
      maxfromjointl t  distance of maximum load from jointl in feet
      mlnfromjointl t  distance of minimum load from jointl in feet)
```

```
FORCECONDITION • RECORD

        eondltionnumber    t  REAL)

        <pointer  variables)

        nextforceconditlon  t  FORCECONDITIONPTRI
        livtloadforce       t FORCEpTRy
        deadloadforce       t FORCEpTRy
        windloadforct       i FORCEpTRy

(eonditlonnumber    t  a numbtr Indicating which loading condition
 nextforeecondition t  tht ntxt foree condition
 »loadforee         t  the Induced *ioadforees in the member)
```

```
FORCE • RECORD

        axlalforee    ! REAL;
        momentjolntl  ! REALy
        shearjolntl   ! REALy
        momentjoint2  ! REALy
        •hearjoint2   ! REALy
        ENDy

{axlalforee   i compression is negative* tension is positive, in kips
 momentjoint1 i amount of applied moment at endjolntl in foot-klps
                positive in counterclockwise direction
 shearjolntl  i amount of applied shear in kips
                positive in upward and rightward directions
 momentjolnt2 i amount of applied moment at endjolnt2 in foot-kips
                positive in counterclockwise direction
 •hearjoint2  I amount of applied Shear in kips
                positive in upward and rightward direstlons

 Note         i this record contains the values of the APPLIED
                forces on the member as determined by the
                ANALYZE application program)
```

```
SHAPE a RECORD

        <pelnt«r variables)

        firstmemberthiskind | MEMBERPTRj

        {variant)
        kind                 I SECTlONTYPEl
        CASE  sectlontype  OF
                WF | (wldt  I HIDEPLANGEPTR)!
                C  : (Chan  I CHANNELPTR);
                A  : (anal  I ANGLEPTR)!
                T  : Cteee  I TEEPTR)!
        END|

<£irstm«mberthis)clnd t linked list of members comprising this section
 kind                 I sectiontype this member is chosen to be
 CASE                 t record containing specific details about seetlontype)
```

```
WIDEFLANGE • RECORD

        designation        i  NAMEf

        {detailing information)

        width              :  REAL»
        depth              :  REALI
        thickness          :  REAL)

        {designing information)

        area               : REALf
        momentofinertlax   : REAL!
        seetlonmodulusx    : REALf
        momentofinertlay   : REAL}
        seetionfodulusy    : REALf
        torsionaleonstant  : REALf
        plasticmodulusx    : REALf
                           : REALf
        plastlemodulusy

        {pointer variables)
                           : WXDEFLANGEPTRf
        nextwideflange     : WIDEFLANGEPTRf
        nextlargerS        : WIDEFLANGEPTRf
        nextlargerA        : WIDEPLANGEPRTf
        nextlargerw        : WIDEFLANGEPTRf
        nextsmallers       : WIDEPLANGEPTRf
        nextsmallerA       : WIDEFLANGEPTRf
        nextsmallerw
<nextwldeflange   : next in linked list of wideflanges
 nextlargerS      : wideflange with next higher seetlon modulus
 nextlargerA      : wideflange with next higher area
 nextlargerw      : wldeflange with next higher weight
 nextsmailerS     : wideflange with next lower seetion modulus
 nextsmallerA     : wldeflange with next lower area
 nextsmallerw     : wldeflange with next lower weight)
```

```
        CHANNEL • RECORD
                (similar to wldeflangt)
                END}
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
        ANGLE • RECORD
                {•lmllar to wldtflangt)
                END)
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
        TEE * RECORD
                (similar to wldtflange)
                END)
```

```
VAR

        headframe       : WIDEFLANGEPTR;
        headshape       : SHAPEPTR;
        headwideflange  : WIDEFLANGEPTR;
        headchannel     : CHANNELPTR;
        headangle       : ANGLEPTR;
        headtee         : TEEPTR;

    {head* : pointer to first record in linked list of * records}

    {Note  : these variables are essential to allow DML procedures
             access to the data item records available for system
             access}
```

```
 <XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX>
{                                                                }
<     Function riNDFRAME takes the Identification of a          }
{ frame, finds tht frama in tha linked list of frames,           }
< and return! a pointer to the frame•                            }
<                                                                }
 <XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX>

FUNCTION flndframeCframeld i INTEGER) I FRAMEPTRf

VAR
        frame i FRAMEPTRI

BEGIN
        frame is headframef
        WHILE frame*.franeld <> frameld DO
                frame i« frame*,nextframe»
        IF frame • NIL
                THEN writelnCTTY, 'That frame doei not exist')
                ELSE findframe i« frame
END|
<xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx>
<                                                        >
<      Function riNDBLANK takes the identification of a >
< "blank", finds the blank in the linked list of blanks,)
< and returns a pointer to that blank. That ls# it      >
< returns a pointer to an objeet of type blank where    >
{ blank can represent any of story, member* joint* ttc. >
{                                                        >

<xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)
FUNCTION flndblankCframeld, blankld t INTEGER) i BLANKPTRf
VAR
        blank t BLANKPTR*
        frame i FRAMEPTRf

BEGIN
        frame !• flndframeCframeld)*
        blank t« trame",blanksi
        WHILE blank",blankld <> blankld DO
                blank s« blank*,nextblanki
        ir blank • NIL
                THEN vrltelnCTTY* 'That blank does not exist*>t
                ELSE flndblank i» blank
END)
```

```
(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx:
{
<      Proctdurt TOPOLOGYALGORITHMi takes a frame
< identification and finds and prints out tha iden-       ,
< tlfleatlon of all tha storys* bays, members* and        }
< joints in tha frame,                                    }
{                                                          }
(xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx)

PROCEDURE topologyaigorithmlCframaid i INTEGER);

VAR
        frama  i1 FRAHEPTRf
        story  i1 3T0RYPTRJ
        bay  i1 BAYPTR)
        member 11 MEMBERPTR}
        joint  i1 JOINTPTRf

BEGIN
        frama i« findfraneCframeld)!

        story i« frana'.storysi
        WHILE story <> NIL 00
                BEGIN
                writalnCTTY, story*,storyid)»
                story t« story'.naxtstory
                END}

        bay *• fra«a*abaysf
        NHILE bay <> NIL DO
                BEGIN
                vrltalnCTTYf  bay*.bayld)f
                bay »« bay'.naxtbay
                ENDf

        member i" frame*.members I
        WHILE member <> NIL DO
                BEGIN
                vrltalnCTTYf  member".aemberld)!
                member  i« member*.nextmember
                END}

        joint  t« frame",jointsf
        WHILE joint <> NIL DO
                BEGIN
                writeln(TTY#  joinf.jointid)i
                joint  i« joint*.naxtjoint
                END?

END}
```