A SYSTEMS DEVELOPMENT METHODOLOGY BASED UPON AN ACTIVE DATA DICTIONARY/DIRECTORY

T. M. Curtis
Quasar Systems Ltd.
March 1981

1. Introduction
2. Historical Perspective
3. Proposed Approach
4. Method and Technique
5. Obstacles to Implementation
6. Conclusions

Introduction

Computers are not very tolerant of humankind communications. Phrases such as

"you know"

"things"

"what do you call it"

"etc."

are incomprehensible to the average COBOL compiler. Similarly people are not tolerant of the machine's "pickiness" and need for detail. As a result of this communications gap, the EDP professional has leapt into the breach to become a translator between the two uncompromising camps, translating the needs of the user into language and terms that may be manipulated by the computer as well as explaining the strengths and limitations of the machines to unsophisticated users.

The problem with this approach is that the translator has become the key element in the cycle. All communications dealing with the development of computer systems must pass through the EDP professional in both directions.

In recent years there have been dramatic increases in the demand for automated systems and the power of machines to provide services.

However, the supply of EDP professionals (translators) has not kept pace with either the demand for

services nor the hardware capacity to deliver the required services. As a result, the limitation on fully utilizing the new hardware power to address the burgeoning demand is us, the EDP professionals.

The traditional approach taken to solve this problem has been to increase the productivity of the EDP professional. A procession of analytical, design and programming techniques has been combined with more powerful languages, data management systems and utility software to address the EDP productivity problem. Although these facilities are worthwhile in their own right, they are merely treating the symptoms rather than the problem.

Our challenge is to develop more sophisticated tools for computers and to raise the level of technical literacy of their users so that they may directly interact with the computer for "routine" development processes. This is a natural continuation of the process that has relieved EDP organizations of the burden of data preparation and entry by using hardware to switch from card input (controlled by EDP organizations) to direct data entry using DDP and on-site terminals. That is, we have turned over operational control of systems to the user. The next evolutionary step is to return routine development tasks to the user.

The problem of increasing the level of technical literacy within our society must be left to our educational systems.

This paper will address the opportunity presented by the need to support direct user/computer communications to effect the development of automated data processing systems.

Historical Perspective

Although each of us may follow slightly different analytical, design and development methodologies, the underlying principle is the same:

--determine requirements

--design a computer system that will satisfy the requirements

--develop the system

The requirements are usually, or should be, phrased in non computer oriented language, comprehensible to the user. These requirements are then transformed into a computer design, and the functions and data are translated into process descriptions.

We have traditionally organized our approach to preparing detailed procedures (programs) into data and processing specifications. The file structures, record layouts and field descriptions are prepared. The programmer must then combine these data descriptions with the procedural process descriptions in a program.

We know from experience that the result has been large, unwieldy, incomprehensible, and unmaintainable programs and systems. To overcome this problem we have adopted structured techniques that stringently define the domain of a process and the size of the resulting module. This is essentially an attempt to limit the number of variables and levels of data and processes the programmer must concurrently deal with.

By limiting the size and complexity of modules we have been able to keep the entire complex of data and processes within the intellectual grasp of the programmer. The result of this structured technique has been to achieve greater productivity and dependability through simplifying and standardizing the fundamental system building block, the module.

However, these techniques do not produce the increases in productivity necessary to respond to current and projected demands.

Proposed Approach

General

The dramatic increase in the power of computing hardware coupled with the relative decrease in cost provides us with the basis of a solution: if we can divert some of the power of the machines from "getting the work done" to easing the man machine interface we can reduce the comprehension gap between users and machines. This approach has previously not been feasible because of the cost of machine power necessary to support this level of interface as well as the requirement to get the job done, ie., application systems required all available cycles.

Theoretical Framework

All systems are composed of two elemental items:

a) an entity

b) a relationship

It is possible to comprehensively describe a system in terms of the component entities and relationships that make up that system.

Definitions:

Entity: "Thing's existence as opposed to its qualities or relations."

We have problems manipulating concepts on a machine, therefore for our purposes an entity may be considered to be;

: That characteristic of something that identifies, describes or quantifies it.

Relation: "What one person or thing has to do with another."

for our purposes a relation may be considered to be;

: A characteristic or series of characteristics that establishes a link between entities based upon some common identity, description or value.

There are three types of entities that can be used to describe the nodes of a system:

a) data

b) processes

c) users

Data, or course, refers to the information maintained, manipulated or produced by the system.

Processes refers to the rules, precedences, time sequences and operations to be performed on the data handled by the system.

Users refers to the owners, users, controllers and authorities responsible for and involved with the system.

Each of these entity types may be further subclassified as follows.

ENTITIES maintained by DD/D

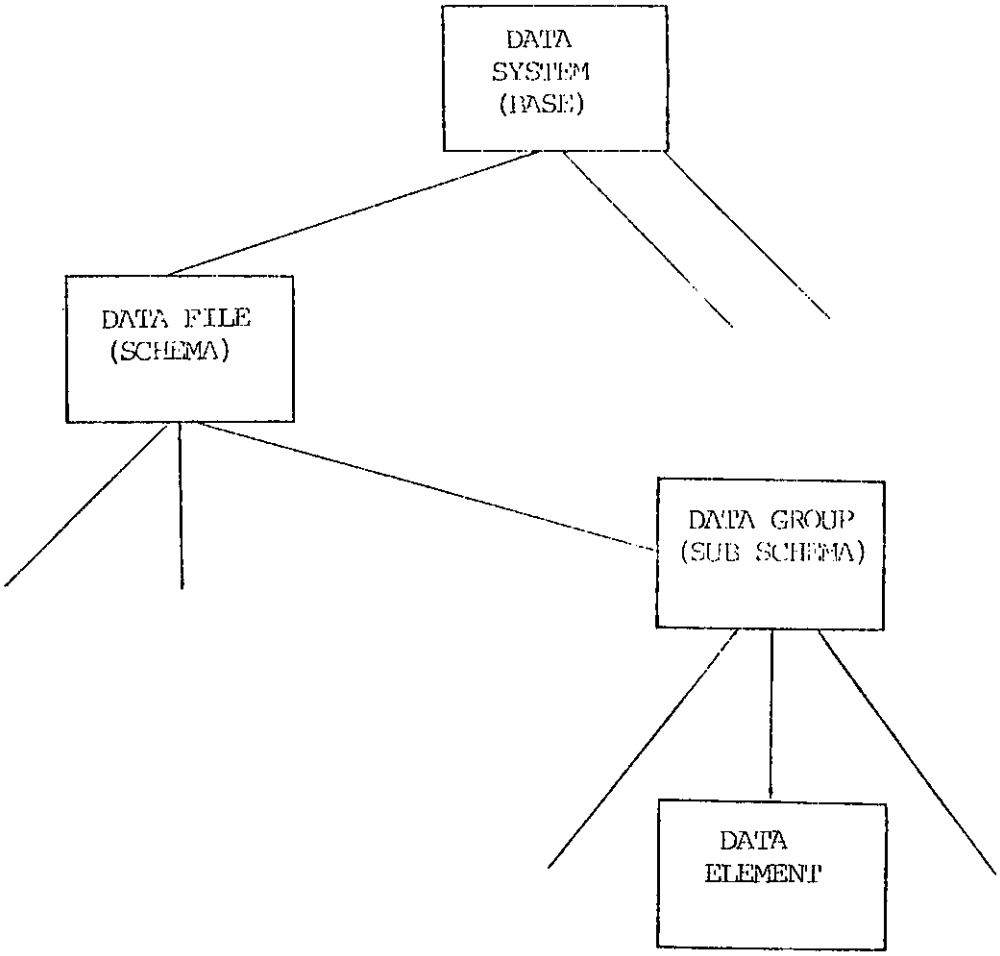
DATA

- Data item - a primitive - data definition
- Data group - sub schema - (record) 1st order assoc.
- Data file - schema - 2nd order association
- Data system - (base) - 3rd order association

PROCESSES

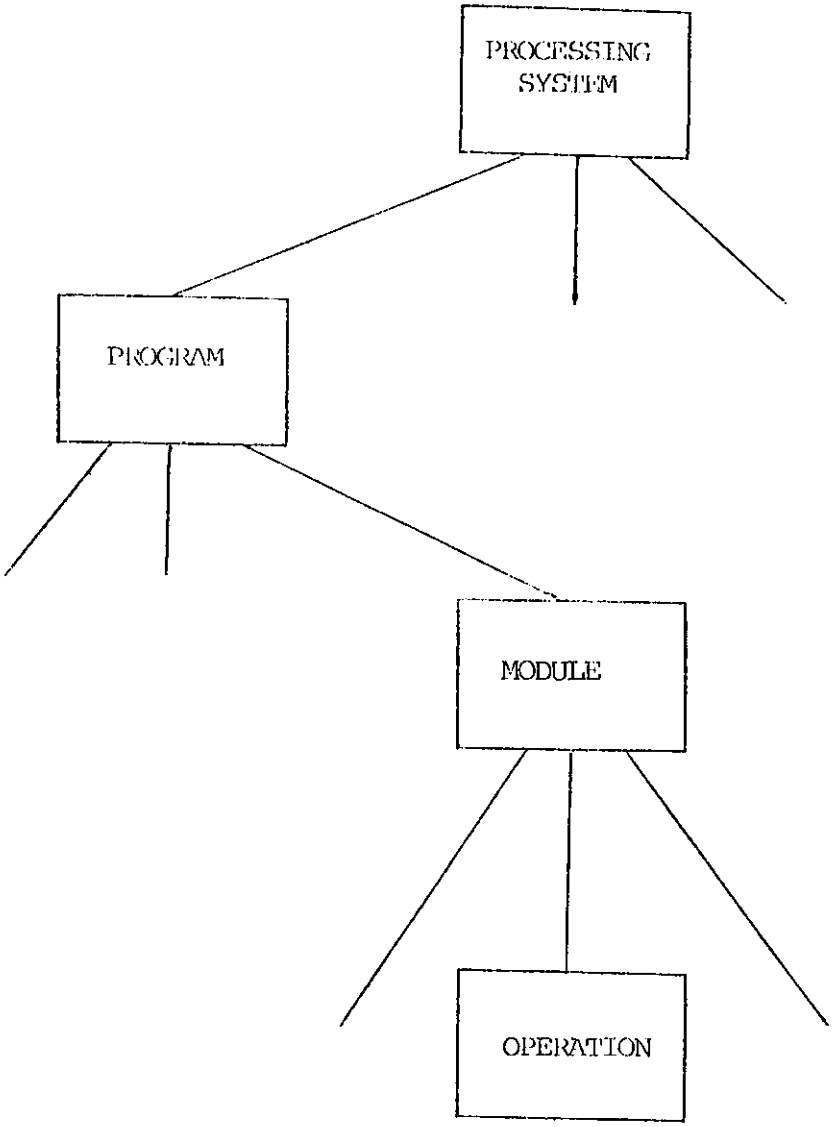
- Operation - a primitive - "+" "-" etc. QUIZ & QUICK commands
- module - an association of functions
- program - an association of modules
- system - an association of programs

A physical hierarchical view of the entity types is as follows:



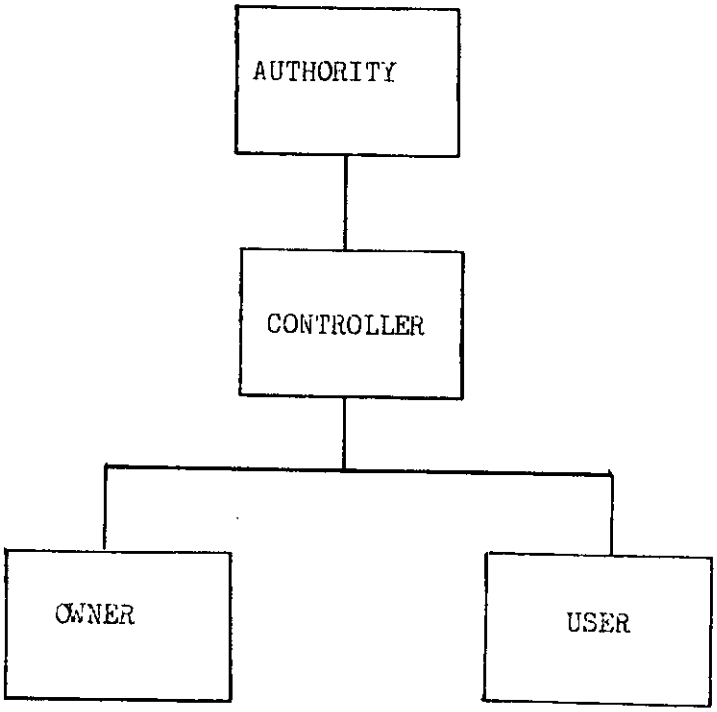
DATA HIERARCHY

Fig. 1



PROCESS HIERARCHY

Fig. 2



USER HIERARCHY

Fig. 3

USER

- Owner - person or process responsible for accuracy and timeliness of value
- User - person or process that "uses" the data
- Controller - person responsible for controlling access to the data or process
- Authority - person responsible for the definition of the entity

Relationships between entities may be of two categories and take one of three forms.

Relational categories are:

a) absolute: the relation between the associated entities exists at all times and under all conditions.

b) conditional: the relationship between associated entities does or does not exist based upon the value of another entity or the result of another relationship.

The three forms of a relationship are:

a) Relative

b) Associative

c) Algorithmic

a) Relative: "What one person or thing has to do with another."

"Kind of connection, correspondence, contrast or feeling that prevails between two persons or things."

for our purposes we will consider a Relative Relationships to be;

: A grouping of entities that collectively identify, describe or quantify a higher level entity.

e.g., all information maintained on an employee is related and provides an identification, description and "value" for that employee.

b) Associative: combine for common purpose

: connection between related ideas

: thing connected with another

for our purposes we will consider an associative relationship to be;

: A grouping of entities based upon a common or related value of individual or grouped elements.

e.g., all personnel working in the products office are "associated" entities.

c) Algorithmic: process or rules for calculation

for our purposes we will consider an algorithmic relationship to be;

: A procedural relationship established between entities with the purpose of identifying, describing, quantifying or deriving another entity.

e.g., the entity "net pay" may be derived algorithmically as Gross Pay minus Total Deductions.

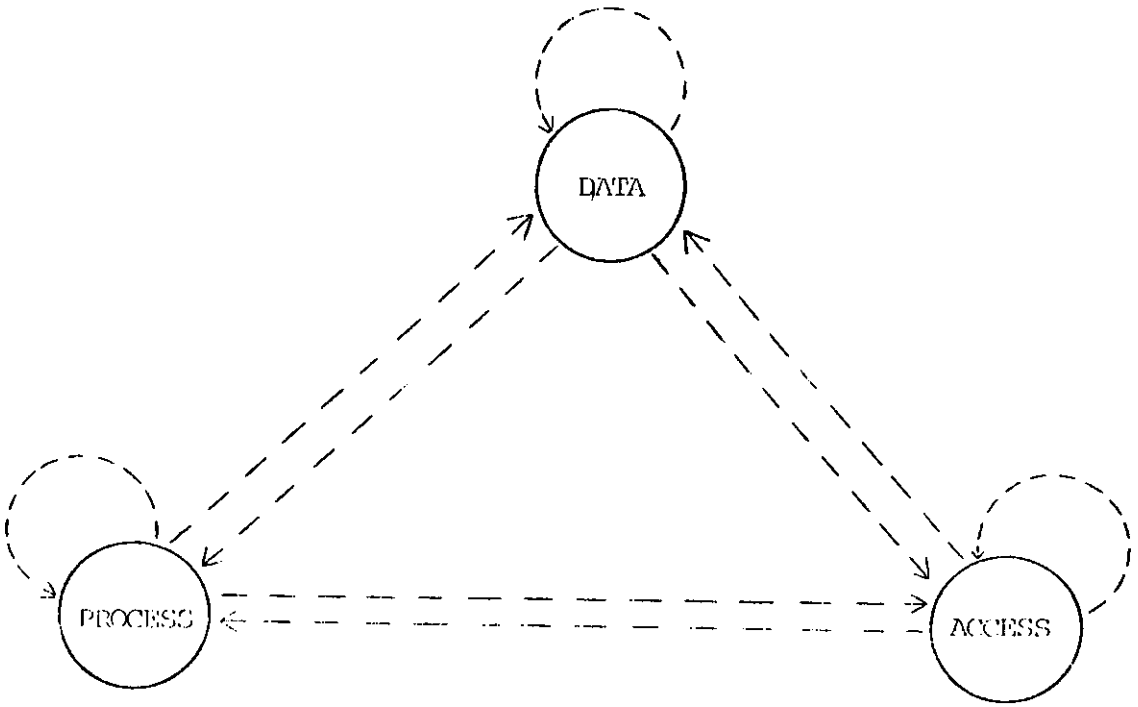
Relationships may be established (may exist):

a) between like entities

b) between dissimilar entities

c) both like and dissimilar entities concurrently

d) recursive (a part may itself be composed of parts, etc.)



RELATIONSHIPS BETWEEN ENTITIES

Fig. 3a

Method and Techniques

The active Data Dictionary/Directory appears to be a viable tool to implement an entity/relationship description of a computerized system. We are all familiar with a number of passive data dictionaries used basically for documentation and data structure source language generation. Packages exhibiting these characteristics have been on the market for years. More recently some dictionaries have become more active, actually resolving references to stored data entities.

The passive Data Dictionary/Directory has the typical structure shown in figure 4. Of course, most current data dictionaries do not maintain process descriptions below the compile unit level, typically a program or subroutine. This structure is not conducive to efficient or effective handling of entity/relationship descriptions. A proposed structure for an active Data Dictionary/Directory is provided in figure 5.

TYPICAL DD/D STRUCTURE

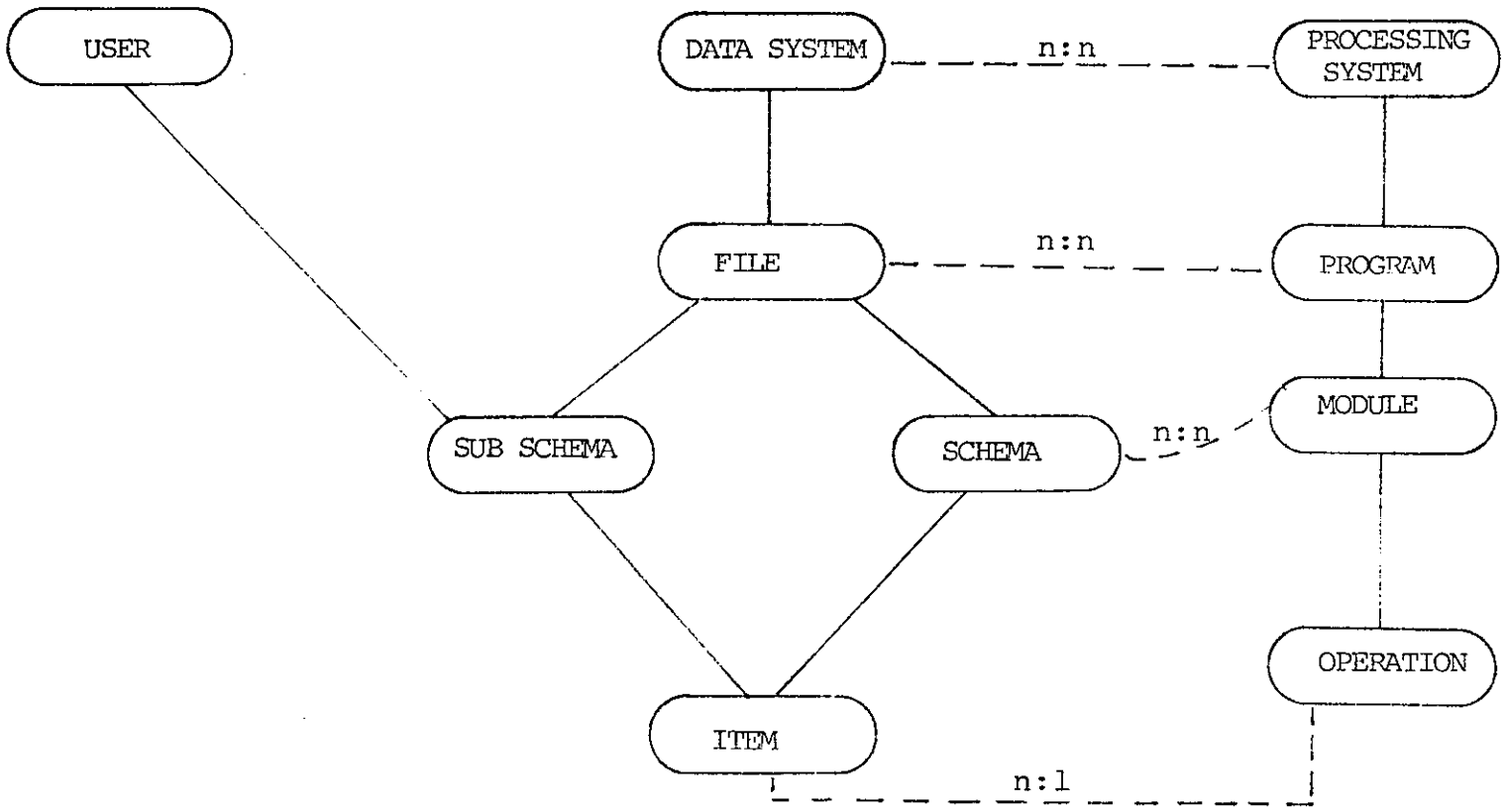


Fig. 4

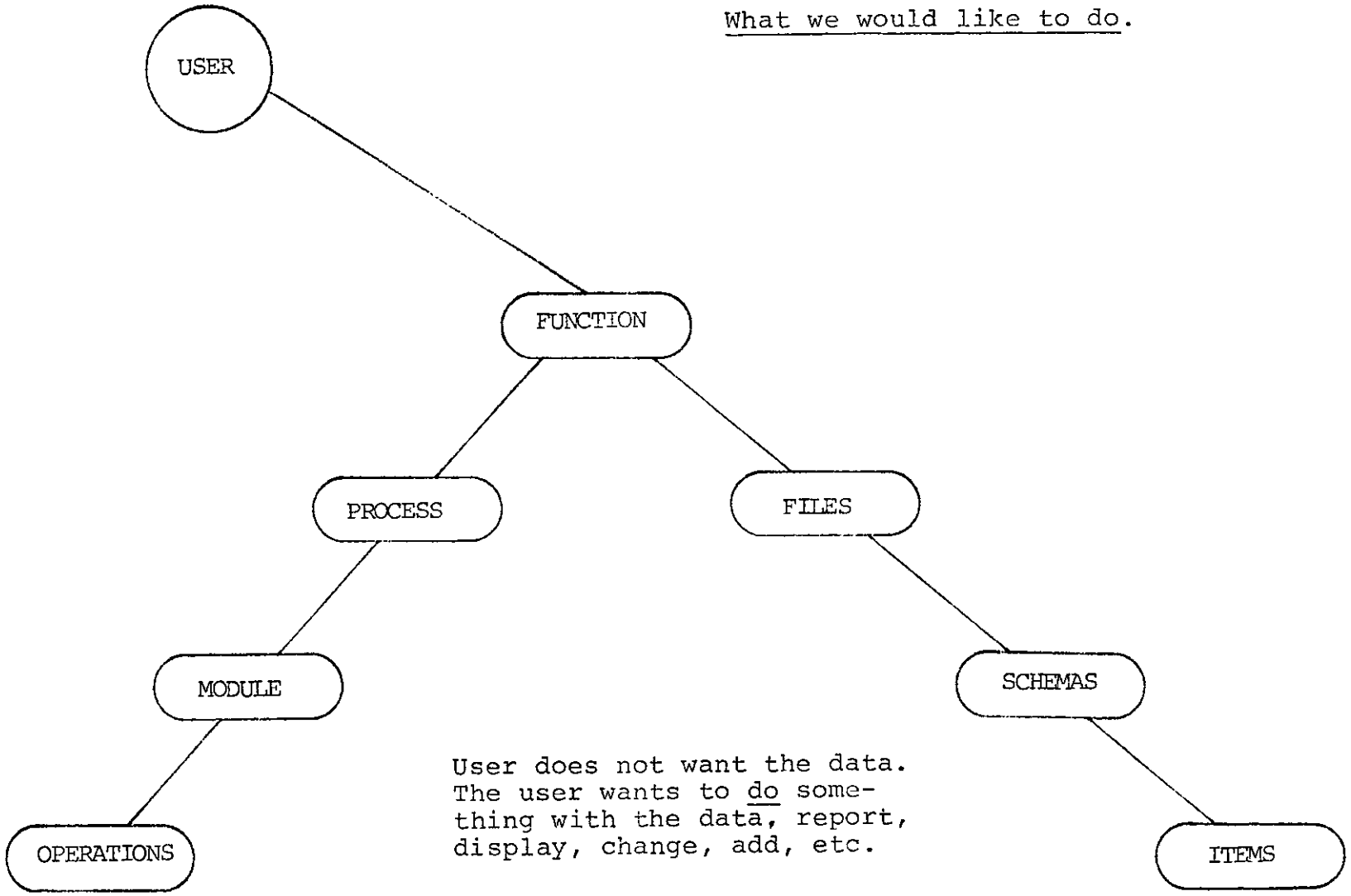
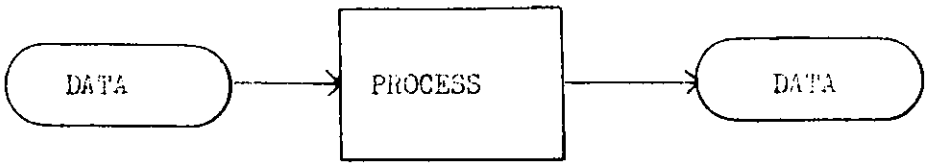

What we would like to do.
USER
FUNCTION
PROCESS
FILES
MODULE
SCHEMAS
OPERATIONS
ITEMS
User does not want the data.
The user wants to do some-
thing with the data, report,
display, change, add, etc.

Fig. 5

The structure may be interpreted such that a series of hierarchical processes and data entities form a set joined by relationships. This set of entity/relationships has been organized and termed a function that is "owned" or "used" by a user entity. Therefore, to use structured terminology, if we can establish and define data/process relationships at each level of the hierarchy we wil be able to describe a system. If this description can then be maintained and manipulated by an active Data Dictionary/Directory, we will have established a Function Processor.

A very simple example of this concept is shown by:



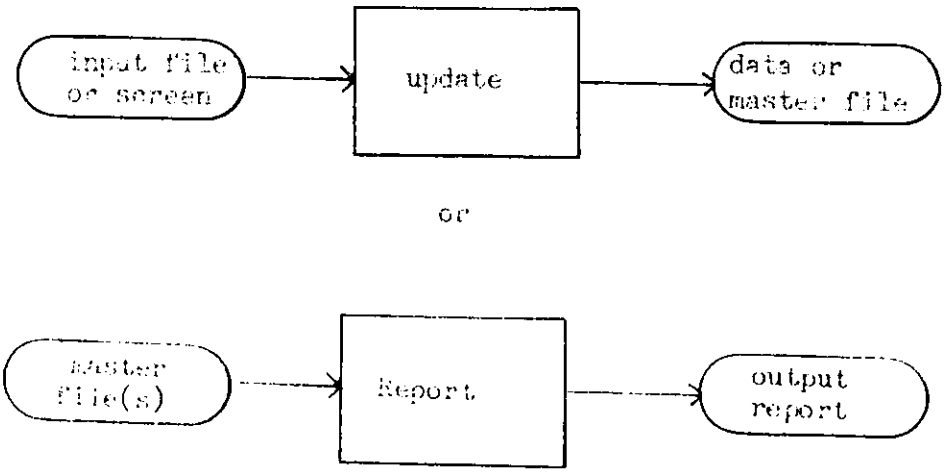
Data can, of course, be a structure

- input file (screen, etc.)
- output report (file, etc.).

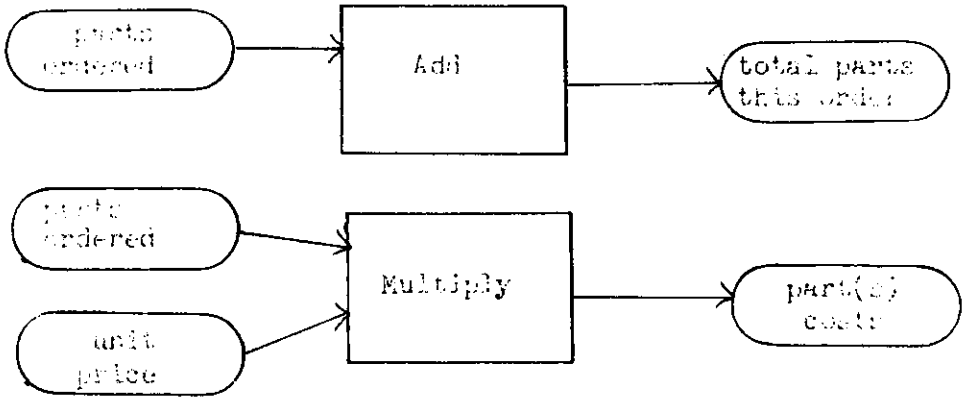
Process can be

- move
- add
- display, etc.

It is possible to describe processes as a series (time sequence) of such entities. At higher levels the description takes the form:



At a lower level, the process/data relationship may be described as:



The best way of describing how this approach may work is through the use of an example. An order entry application has been selected for demonstration purposes. The processes to be performed are:

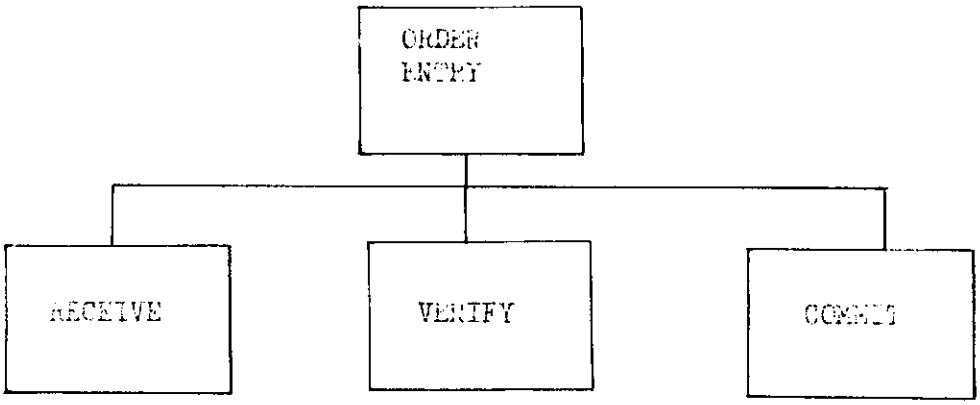
a)	Receive order	(Receive)
b)	Perform verification and credit checks	(Verify)
c)	Commit stock from inventory	(Commit)
d)	Back order "shorts"	(Back Order)

e) Print picking slips (Pick slips)

f) Generate invoice (Invoice)

for purposes of this discussion we will consider Back Orders, Picking Slips and Invoices as products of Commitment.

We may therefore describe the process hierarchy as:



The corresponding data entities are:

a) Orders

b) Client file

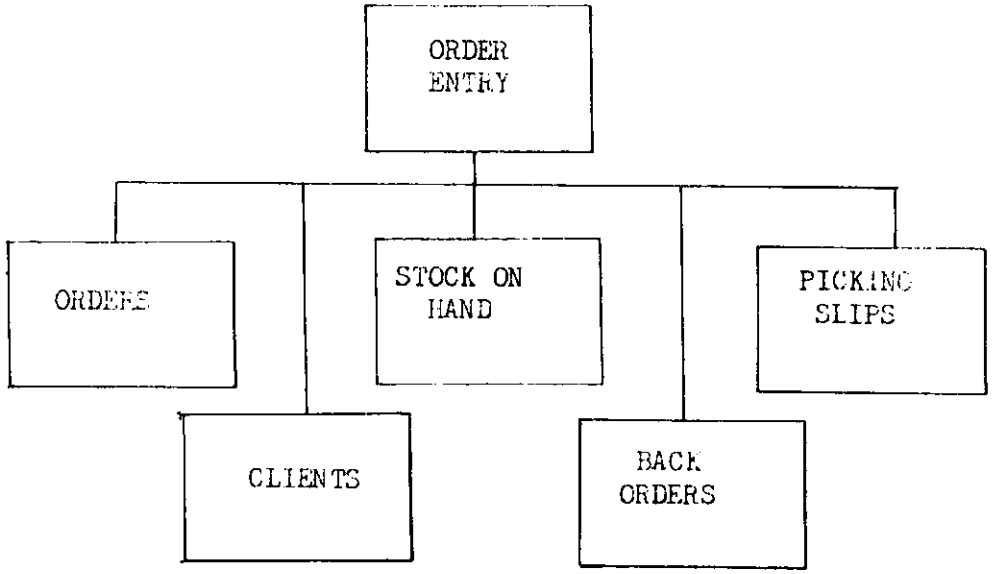
c) Stock on hand file (INVENTORY)

d) Back Orders

e) Picking Slips

f) Invoices

The data hierarchy may therefore be represented as:



It should be pointed out that the association of a data entity to a process entity does not imply ownership. Rather, the relationship may be classified as:

a) use or input

b) update

c) create

d) delete

e) derive

We may define this function as

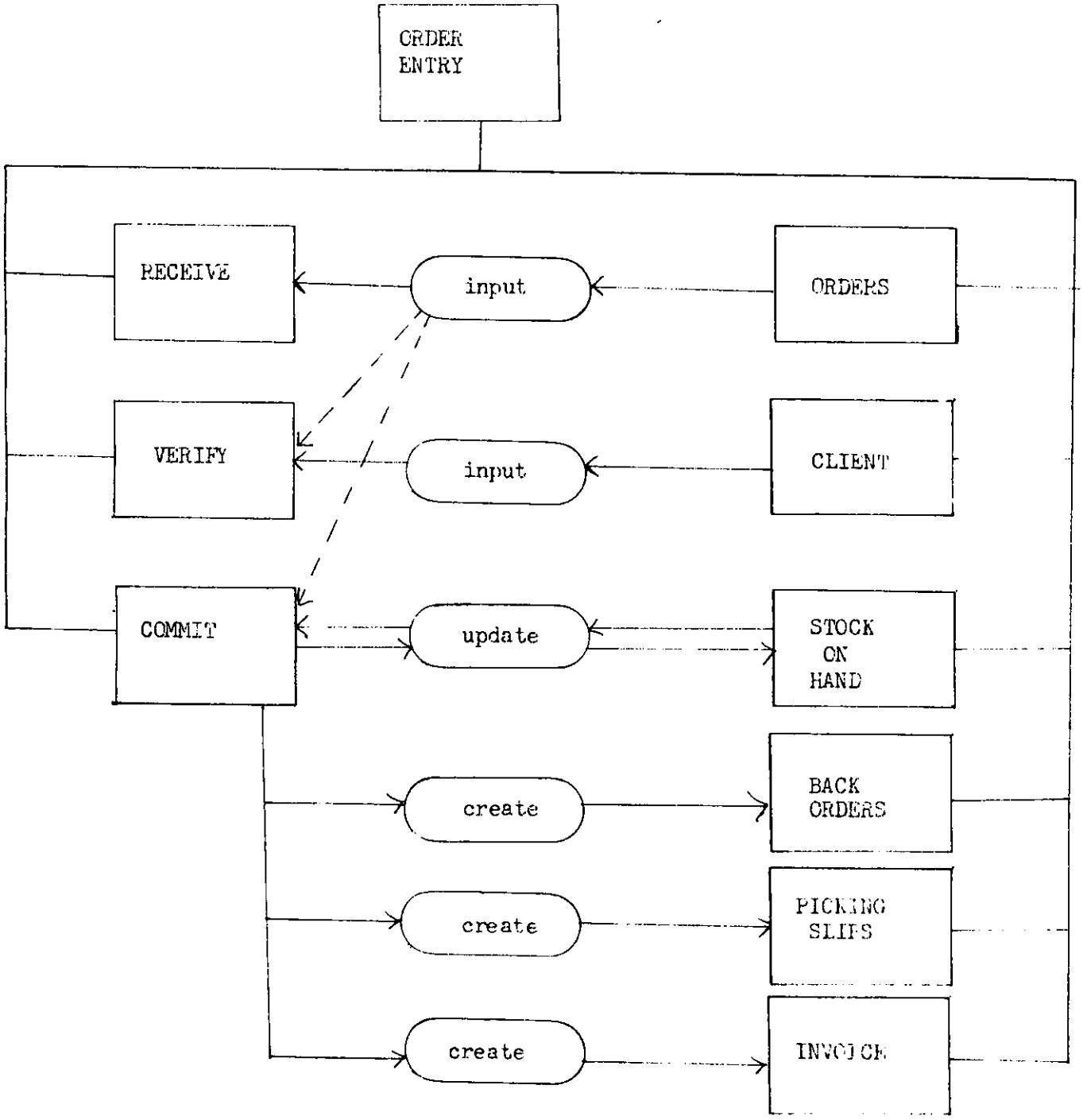
Function: ORDER ENTRY

PROCESS	DATA ENTITY USED	USE	AVG. VOL.	DISTRB	PEAK
RECEIVE	ORDERS	INPUT	Information to be used to support the structure design.		
VERIFY	ORDERS	INPUT			
	CLIENTS	INPUT			
COMMIT	ORDERS	INPUT			
	STOCK	UPDATE			
	BACK ORDERS	CREATE			
	PICK SLIP	CREATE			
	INVOICE	CREATE			

The RECEIVE process is straightforward and does not have to be elaborated upon here.

The VERIFY process is interesting. In essence VERIFY is meant to apply the validation rules that are part of the data definition for each item in the source structure. A sample source structure, in this case an order, is presented below.

```
ORDER NUMBER
CLIENT IDENTIFIER
CLIENT ADDRESS
CLIENT CONTACT NAME
CLIENT CONTACT PHONE
SALESPERSONS IDENTIFIER
DATE
ORDER LINE
            PART NUMBER
            PART DESCRIPTION
            PART UNIT PRICE
            LINE EXTENSION
SALES TAX
TOTAL
```



ORDER ENTRY ENTITY RELATIONSHIPS

Fig. 6

Each data item of the structure is defined as a data element within the dictionary.

Typical definitions will contain:

- unique name
- synonyms
- description and purpose
- type of data
- edit/validation rule(s), severity and messages
- source or derivation
- principal site
- responsibility
- authority
- security restrictions
- links to processes
- links to other data items.

Once the relationship between the process (VERIFY) and the data entity (ORDER) has been established the functional processor may then sweep the data structure applying the edit/validation rules. These rules are not limited to range and type checks but can reference other data items described in the dictionary. As an example, the verification of the CLIENT IDENTIFIER may involve the application of a number of rules.

a) Type is numeric

b) Range 000 - 999

c) Registered on client file CLIENT FILE: PRESENT

d) CREDIT equal OK

The first two rules are simple checks. The third and fourth rules require the resolution of data entities contained in other (but related) data structures. The projected operations of the Function Processor in resolving these references would be:

a) retrieve definition for CLIENT IDENTIFIER

b) resolve "immediate" rules

d) determine other entities required: CLIENT FILE: CREDIT

e) resolve physical location

f) obtain physical representation (record)

g) apply rule(s)

h) set status, produce message, etc.

As this example has illustrated, it is possible to perform the VERIFY function by invoking a primitive operation (VERIFY) and relying upon the data and process specifications maintained by the Data Dictionary/Directory.

Similarly the COMMIT process may be simplified to four subprocesses:

- update the stock on hand
- create back orders
- create picking slips
- create invoices

Again the data and process structures used to

perform these operations may be defined in the active data dictionary/directory. The relationships between origin data structure, process and target data structure are given by the structure and linkages of the dictionary.

Utilizing a top down approach to system specification we are able to comprehensively describe the application in a hierarchical fashion.

Once we have the hierarchy of functional blocks we may further decompose the problem until the leaves of our hierarchical tree represent primitives in terms of data and process entities. Anyone familiar with the Jackson methodology will be acquainted with the technique and representation of processes and data.

- Data structures are represented hierarchically.
- Relationships are expressed as correspondences.
- Processes are "operations lists" and are merged with the consolidated data structures, ie., hierarchically structured.

This organization can be maintained by a data dictionary. Indeed many data dictionaries already maintain most of the information necessary to support this type of process/data description.

Obstacles to feasible and practical implementations.

There are two major obstacles to a feasible implementation of this model. The principal difficulty is the development of a non procedural grammar that is concurrently

a) natural language like - to allow the end user to specify data and processes in familiar terms

b) structured enough to provide comprehensive and unambiguous data and process descriptions to the system.

The second obstacle deals with the operating efficiency of such a system. The hierarchical trees of entities and their relationships can quickly become extremely large and complex for even medium sized applications. The organization, maintenance and reference of such a structure will require considerable sophistication to provide the responsiveness, performance and reliability necessary to produce a workable tool.

Conclusion

The techniques and approach outlined in this paper depend upon tools and technology currently available. What is necessary is the impetus required to revise our thinking about how we specify, design and develop computerized systems. The challenge of closing the gap between the unsophisticated user and the uncompromising computer can be addressed from either end. This approach attempts to use the power of the computer to accept non procedural, non technical descriptions of functions, data and processes, and generate automated systems.