308

A SURVEY OF FORMS PROCESSING TECHNIQUES

By

RAJIV KAPOOR

MBA, KANSAS STATE UNIVERSITY, 1982

A MASTER'S REPORT

Submitted in partial fulfilment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

- 1984 -

Approved by:

[illegible]

Major Professor

ACKNOWLEDGEMENTS

I would like to take this opportunity to acknowledge several individuals who have been instrumental in the preparation of this report. Academically, I would like to express my sincere appreciation to my committee members, Professors Paul Fisher, Virgil Wallentine and Rod Bates for their counsel and recommendations. I wish to specially thank my major professor, Dr. Fisher, for the many hours he must have spent in editing my report and also because he was so patient and understanding of my circumstances of trying to do my report while teaching at Missouri Southern State College. I draw inspiration from him in the tireless way he performs his work. It was he who earmarked the extensive literature survey I had to do. I also wish to express my gratitude to him for always providing encouragment,and for appreciating my dual commitment to finishing my degree and teaching Computer Science at Missouri Southern. I wish to place on record my deep appreciation of the Department of Computer Science at Kansas State University for giving me the tools to be able to do well in the exciting field of Computer Science. I have always found the environs of Fairchild Hall a friendly place to work. I also wish to thank the Department of Computer Science at Missouri Southern State College for encouraging me.

Personally, I wish to thank my wife Neena, who has always helped me through all of my endeavours.

INDEX

LIST OF FIGURES

Chapter 1

INTRODUCTION

The purpose of this research is to investigate the role of utilities; in this case forms; in supporting access to data bases dispersed throughout a network. An amplification of this objective is as follows:

The objective of this research is concerned with the development and use of forms as a design specification for network utilities which allow access through a common query facility to data at any node of a decentralized network of databases with non-homogenous data models, non-homogenous data model implementations and non-homogenous database management systems (DBMS) query facilities.

1.1 OVERVIEW SOLUTION

In the environment of data access there are several associated problems which are listed below. With regard to these problems some are amenable to solution or resolution through forms based systems.

. Identification of the data in the network -- the solution of this will be handled through the use of data dictionary technology.

. Location of the data in the network -- the solution can be developed from an extension of data dictionary technology to directories making available an integrated data dictionary/directory technology.

. Translation of request for data at one node into a synonymous request for data at a different node. This entails the following consideration :

- Resolution of potential naming discrepancies - this problem involves synonyms and homonyms; its solution can be affected through use of the data dictionary.
- Reconciliation of potential data model discrepancies- this is a problem of mapping between the hierarchical, network and the relational models whose solution involves forms and metadata available in a data dictionary.
- Differences in query languages - this is a language problem; its solution involves the translation from global query language statements into primitives which implement the semantics of the initial query in terms of the data model at the node(s) where the data resides. The metadata required for the solution of this problem resides in the integrated data dictionary/directory; the solution involves the design of a global user interface, of the primitives and the translation algorithm.

A forms based programming language can be applied to the above problem. A form is the same for each node in the decentralized network of databases with non-homogenous data models and non-homogenous data model implementations, and non-homogenous database management system (DBMS) query facilities. If these nodes could communicate through the common language of forms, then any such interface problems could be resolved. A form presents an easily understood context for an element, and is virtually free of specific relationships.

1.2 DICTIONARY SYSTEMS

Current Data Dictionary Systems (DICS) address the problem of locating data and processes from a logical point of view ie. they contain metadata that shows the usage of data and proccesses by other data and processes. In a distributed environment, it is additionally neccessary to identify the physical location of data and processes : the facility that is used to handle this problem is a Directory System (DIRS). It might be a good idea to combine the Data Dictionary System (DICS) with the Directory System resulting in an integrated Data Dictionary/Directory (DIRS/DICS) system. Advantages of this are:

. redundancy between DICS and the physical locator system DIRS is removed.

. The same technology that was applied to a DICS in a distributed environment can now be applied to the integrated DIRS/DICS system.

The Integrated DIRS/DICS System may resolve the following problems:

. Control of distributed data and metadata.

. Translation of "global queries" into meaningful local queries.

1.3 CONTROL OF DISTRIBUTED DATA AND METADATA

As databases are distributed, the need for distributing data also varies, as some of this data is required at the various nodes to allow processing at these points. Thus, though a single node contains the integrated DIRS/DICS the need for distributing data for operational reasons introduces redundancy. The DD/DS may be distributed functionally across the network for performance or reliability reasons. Two kinds of problems arise:

. The problem of synchronization of distributed data especially in the case where two users at two different nodes in a distributed database are vying for access to the same data simultaneously.

. Problems concerning control of distributed data depending upon where the DIRS/DICS should reside.

It becomes neccessary to have software which controls theinterface to the DIRS/DICS for the purpose of insuring that the neccessary data for control and access is communicated between the various systems. This control and access software is one of the utilities which has been a focus of extensive research.

If you have data distributed at the nodes and also have an integrated DIRS/DICS system then in actuality the same data is maintained in two places leading to problems of synchronization.

If however, you combine both kinds of data into one DIRS/DICS then though the problem of synchronization gets solved, problems of bottlenecks are introduced when two independent users would have difficulty in having queries answered about or with the same data at the same time.

The solution to these problems lie in the design and development of a new architecture of DIRS/DICS to support the management and control functions of the integrated directory/dictionary and the data.

Where data is to be distributed, several issues have to be resolved:

1. Is there to be "master dictionary" at one node and "slave dictionaries" at all other nodes?

2. Is there to be a single directory for the entire network?

3. Is a single dictionary to be replicated at each node of the network?

In answering these questions we will consider three possibilities:

a) one centralized directory covering all distributed centers.
b) one directory at each node and no central mechanism.
c) one central master directory and local subsets of the central directory.

A satisfactory form of overall control can only be implemented if options a) or c) are chosen because only in these options is there some central control over data when additions, deletions and access rights to the database are involved. Else, every node in the network could do updates without permission from some sanctioning authority and could lead to an unplanned and haphazard change in the data.. Option b) is the easiest to implement and the most difficult to control and also relies heavily upon communications, since a program that requests data on another machine does not know where the data might be and has to poll every distributed center until it the data is found.

Other solutions to synchronization and bottleneck problems may be to develop a hierarchy of data dictionary systems consisting of a "master" data dictionary system at the top level of the hierarchy

and specialized control or management dictionaries at the second level and DBMS specific data dictionary systems at the third level. A local DBMS is given a default option e.g. enters into a special routine if the data requested by a program is not available locally. This special routine gives access to a higher level of directory built as a separate system, the object of which is to route a data request to the correct machine or reject it on the grounds that it is not available within the network.
One advantage of this is that all remote accesses and updates are thus routed through a point where they can be monitored and controlled. However, this advantage is also a liability with regard to communication stress.

The updating of dictionaries and directories may happen anywhere and at any time. There are some obvious privacy considerations: only database administration staff should be allowed to update and, within that general constraint, different people may be given different priveleges. The propogation of updates should happen in a semi-automatic manner: while a change may be started by a local site, it may be dangerous to accept it as it stands, without some check by a central co-ordinating body. It seems therefore that one needs a facility for incorporating a number of changes, printing or displaying a revised dictionary and directory, obtaining approval, with or without changes, and then propogating automatically the agreed version.

1.4 TRANSLATION OF GLOBAL QUERIES INTO MEANINGFUL LOCAL QUERIES

The usefulness of any database environment is ultimately measured by the user interfaces which exist to support access to the data. The query language interface is one of two basic user interfaces generally available for most database systems (the other being the programming language interface). It provides the user with an easy-to-use language by which ad-hoc queries can be answered in a timely manner. This can be accomplished without the need for the high overhead activities of program coding, debugging, compilation and testing.

While query language syntax varies widely, the architecture of a conventional query language is, for the most part determined by the data model of the database system being used, whether hierarchical, network, or relational.

The fact that the query language syntax varies widely is a major problem in a network of distributed databases. Some query formats are meaningless to other data models; they only have relevance in models with a particular structure. To solve this problem, a normalized query facility/ user interface is required.

A language built around documents would resolve the above problem. Documents serve as an all-purpose media for communicating information. A document environment with spread sheet capabilities and which uses either global or local names provides a readily acceptable format, a transparent mechanism without regard to existing database structures, and a representation which is translatable or mappable into any number of database models. The global and local names will be handled as specified by the DIRS/DICS facility.

Using a forms based query language, it is easy to provide a mapping from the document to the actual query. One can derive programs to satisfy information requests from the document format. In some cases, where the data is distributed over several systems or models, the request may be mapped into a more complex form such as a program.

Chapter II

FORMS PROCESSING

A form is an information holding object consisting of two parts: 1) a form heading which describes form name and form components; and 2) one or more form instances (or form occurrences) conforming to the form heading. A collection of all form instances is known as a file of that form identified by its form name.

2.1 SPECIFICATION OF FORMS PROCESSING (16)

Many authorities consider that forms are the most natural interface between a user and his data. An example of a form is:

(PRODUCT)						
PROD-NO	PNAME	TYPE	(SUPPLIER)	(STORAGE)		PRICE
			VNAME	BIN-NO	LOC	
110	PIPE	PVC	AQUA	B1	SJC	0.79
			CHEMTRON	B2	SJC	
				B3	SFO	
120	PIPE	STEEL	ABC	B4	SFO	4.10
			CHEMTRON			
210	VALVE	STEEL	AQUA	B5	SJC	0.45
			ABC	B6	SFO	
			CHEMTRON			
221	VALVE	COPPER	ABC	B7	SJC	1.25
			CHEMTRON	B8	SFO	
			ROBINSON			

Fig 1. Example of a Form

The form heading assumes a role that is commonly known as data structure definition or schema definition in the data processing community. Its purpose is to formally define the form name, components of the form and structural relationships among components. Fig 2. shows the form heading (in upper case) for a form named PRODUCT. The top line of the form heading contains form name. Components and form structure are represented as follows: Item names are represented in columns. Group names are placed on top of its components. Parentheses are used to denote repeating components. Nesting of repeating groups are represented as levels of parenthesized group names. Parenthesizing the name of the outermost group (i.e. the form name) indicates that the form may have more than one instance. A double line placed at the bottom of the form signals the end of the form heading. A corresponding hierarchy graph for PRODUCT is also included in Fig. 2. It should be obvious that mapping the hierarchical relationships into form heading is straightforward.

```
--------------------------------------------------------------
                    (PRODUCT)
--------------------------------------------------------------
 PROD_NO    PNAME      TYPE      (SUPPLIER)  (STORAGE)   PRICE
                               --------------------------
                                   VNAME      BIN-NO  LOC
==============================================================


            -------------------------------------
            PROD-NO   PNAME    TYPE     PRICE
            -------------------------------------
                                |
                                |
              ----------------------------------
              |                                |
   SUPPLIER   |                     STORAGE    |
        ------------                 --------------------
          VNAME                         BIN-NO     LOC
        ------------                 --------------------
```

FIG. 2   Example of a form heading and its corresponding hierarchy graph

2.2 FORM PROCESS

Each form process is defined as an activity which takes one or more inputs and produces one form as normal output. A big business may have several form processes. Application specialists can decompose a complex procedure into form processes.

2.3 SPECIFICATIONS

In general, specifications of a form process starts with a title line, followed by form heading of the output, with descriptions of data and qualifications for the process. An END is used to indicate end of a specification as shown in Fig. 3.

```
INSERT-NEWPROD:   INSERT INTO PRODUCT

          ---------------------------------------------------------
                              (PRODUCT)
          ---------------------------------------------------------
           PROD-NO    PNAME   TYPE   (SUPPLIER)  (STORAGE)  PRICE
                                       VNAME     BIN-NO  LOC
===================================================================
SOURCE |    *           *       *        *          *      *     *
--------------------------------------------------------------------

END
```

Fig 3. Specification of a form process

2.3.1 Title Line

The first line of specification (Fig. 3)

INSERT-NEWPROD: INSERT INTO PRODUCT

is the title line. It specifies the name of the form process (INSERT-NEWPROD), operation to be performed (INSERT_INTO) and the name of the output form (PRODUCT).

Process name is used to uniquely identify a form process. The name of a form process can be the same as that of the output form as long as no two form processes are given the same name.

2.3.2 Operation

Operation refers to the activity which produces/modifies the form specified in the title line. Many types of operations are possible. They include: CREATE, DERIVE, INSERT, DELETE, UPDATE, PRINT, QUERY & COMPOSE.

CREATE is an operation that constructs a new form or a new file.

DERIVE makes a new form only temporarily. It exists only for the duration of a business procedure.

INSERT inserts form, group or other items into an existing file.

DELETE deletes form, group or item instances from a file.

UPDATE modifies some item values in a file.

QUERY causes contents of a form to be displayed on a terminal.

PRINT causes generation of a hard copy of the form.

COMPOSE means to combine word processing and data processing facilities. It is an operation which allows insertion of data (from a data file) into a text file according to a template.

2.4 DATA AND PROCESS DESCRIPTIONS (16, 17)

There are basically two categories to make the specification of forms complete: characteristics and constraints on data, and qualifications for the operation. The first category specifies the type of each item and neccessary errors to check; the second category provides the neccessary operations for the computer to generate tailor made code.

2.5 DATA CHARACTERISTICS (16, 17)

Data descriptions may include DATA-TYPE, KEY, UNIQUENESS, ORDER, VALUE, NULL-OK, OCCURRENCE, COPY visibility, and COPY-ID. As shown in Fig. 5 below characteristics and constraints on data can be specified in rows under appropriate columns of the form heading. Note that when a `Y' or `N' is specified, they mean `YES' or `NO' respectively. A number enclosed in a pair of angle brackets refers to a footnote. More specifically, <i> at column j refers to a corresponding row of footnote <i> relevant to column j. For example, VALUE OF LOC is described by footnote <3>, which, in turn specifies that it can only be ANY OF (`SAN JOSE', `SAN FRANCISCO', `LOS ANGELES', `NEW YORK', `BOSTON', 3 `?').

DEFINE PRODUCT

	(PRODUCT)						
	PROD-NO	PNAME	TYPE	(SUPPLIER)	(STORAGE)		PRICE
				VNAME	BIN-NO	LOC	
DATA-TYPE	NUM(3)	CHV(6)	CHV(9)	CHV(8)	CHV(4)	<2>	<1>
KEY	Y			Y	Y		
UNIQUENESS	Y			N	Y		
OCCURRENCE	1000			16	10		
ORDER	ASC				ASC		
NULL-OK		Y	Y				
COPY 1 COPY 2				N			
VALUE	GT 100					<3>	
<1>	NUM(7).NUM(2)						
<2>	CHV (15)						
<3>	ANY OF (`SAN JOSE', `SAN FRANCISCO', `LOS ANGELES' `NEW YORK', `BOSTON', 3 `?')						
COPY-ID	COPY 1 = `SALES', COPY 2 = `PURCHASE', COPY 3 = `PRODUCTION', COPY 4 = `DESIGN'						

END

Fig. 4. Desription of data in PRODUCT form

The following are explanations of the material shown in Fig.4:

- DATA TYPE describes the types of items which comprise a form

- KEY denotes an item (or collection of items) whose value uniquely identifies an instance within a group. This information is important for fast accessing and for generating optimized code to carry out an operation.

- UNIQUENESS conveys the fact that every instance of the specified item in the form has a distinct value.

- ORDER specifies the ordering of instances within a form or ordering of group instances within parent instance. Ordering may be ASC (for ascending) or DES (for descending)

- VALUE specification can be used to provide information on validity of data.

- NULL-OK means that a null value is acceptable for that item.

- OCCURRENCE specifies the number of forms instances. This specification is used to provide data volume information, which is useful for estimation of needed storage space. Specified number represents the maximum number of occurrences within a parent instance. The example in Fig.4 shows that there are 1000 instances of PROD_NO in the PRODUCT form.

- COPY VISIBILITY specifies whether to have exact copies of form instances or multi parts (i.e. some instances are blocked off). In other words this determines what the copies will show.

- COPY-ID is the name designated to each copy.

In summary, properties of and constraints on data are specified by means of DATA_TYPE, KEY, UNIQUENESS, ORDER, NULL_OK, OCCURRENCE, COPY VISIBILITY and COPY_ID.

2.6 PROCESS QUALIFICATIONS (16, 17)

The purpose of process qualifications is to provide more specific descriptions of the form process. Qualifications may include information such as the source of data (SOURCE); conditions to be applied for selecting instances from an input for processing (CONDITION); items to be matched when constructing an output instance from two or more input forms (MATCH); retaining or elimination of duplications (ELIMDUP); and deletions effected (DELETION).

2.6.1 SOURCE

Source for the column value specifies how or where to obtain the instances relevant to the operation. For PRINT, QUERY, CREATE, or DERIVE source must be specified for all items of the form. For INSERT and UPDATE source denotes the new instances to be inserted or to be used, as replacements. For DELETE source need not be specified.

There are many ways to specify the source:

1. An asterisk (i.e. `*') under an item indicates that the value is to be supplied (or filled in) from the data base.

2. A form name under one or more components specifies that values of these components are to be obtained from the corresponding components of the specified form.

3. An expression involving arithmetic operations specifies how the new values are to be derived.

4. "CASE" expression allows varying assignment to a particular item.

5. A set of built-in functions can be used for more complex source of data. Built in functions such as SUM, COUNT, MAX, MIN and AVG can be used to specify an aggregate value as the source of data.

6. Set expressions can be applied to homogenous input forms - PLUS, MINUS, UNION, INTERSEC.

CONDITION - the purpose of the CONDITION specification is to provide criteria for selecting instances for processing. Boolean expressions can be used there. In addition, AND and OR also may be used to imply complex conditions.

MATCH specification provides a facility to tie input forms together in a meaningful way by matching some attribute of one form with an attribute of another form. When a match is found, the criterion specified in the CONDITION is applied. When there is no match, no output instance will be produced. However, the "no match" situation may be treated as an error by assigning the unmatched instance of the specified form to an error file. In other words, to tie forms in a meaningful manner we can, in general, use either MATCH or CONDITION specification.

ELIMDUP provides a facility to eliminate duplications in a form.

DELETION specifies what is to be deleted from the form when the specified condition is met.

2.7 SUMMARY OF PROCESS QUALIFICATIONS

The following matrix in Fig.5 summarizes the applicability of qualification at row i to operation at column j. An entry of 'Y' at (i,j) denotes that the qualification i must be specified for operation j. Conversely, 'N' at (i,j) denotes that qualification i is not applicable to operation j. A blank entry means it is optional.

	INSERT	DELETE	UPDATE	CREATE/DERIVE PRINT/QUERY	COMPOSE
SOURCE	Y	N	Y	Y	Y
CONDITION		Y	Y		N
MATCH					N
ELIMDUP					N
DELETION	N	Y	N	N	N

Fig.5 Summary Qualification/Operation Matrix

2.8 INTEGRATING DATA & WORD PROCESSING WITH COMPOSE (16, 17)

COMPOSE is an operation that allows insertion of data (from a data file) into a text file according to a template. This data, from forms which have been updated by the specification constraints and operations described above, can be used to write standardized letters with imbedded data. The text is standardized but the data is procured from the applicable form as shown in Fig. 6.

```
                                   (TODAY)

(CNAME)
(CADDRESS)



Dear Sir:

We wish to acknowledge your order of (DATE). Due to unanticipated
high demand on certain items, we are unable to schedule for
immediate delivery some of the products that you have ordered.
They are listed below.

               PRODUCT                      QUANTITY
               -------                      --------

         (PROD-NO)    (PNAME)               (QTY)

We have placed these items on back-order. Please be assured that
we will not spare any effort to fill your order. We apologise for
the delay and thank you for your patience.



                                  Yours truly,



                                      Rajiv Kapoor
                                      Shipping Department
                                      ABC DISTRIBUTORS
```

Fig. 6 Example of a Text Template

2.9 A BUSINESS PROCEDURE SPECIFICATION LANGUAGE (16)

Thus far a formal language for forms processing has been described. But to enable a business procedure to be automated, we need two additional constructs. One is the specification of dispersement of form instances or messages - ROUTING. Another is the specification of conditions, called triggers, which when satisfied will cause a form process to be executed or routing to be initiated.

So a business procedure definition consists of one or more statements from three major categories of constructs: TRIGGER for specification of conditions; PERFORM for invocation of a form process or other predefined procedure, and ROUTE for the dispersement of messages or forms.

2.10 TRIGGERING (16, 17)

Triggering refers to the conditions for starting up an activity which can either be a form process or routing. An activity can be triggered upon arrival of certain form instance(s) and/or a signal from a 'post', and/or at a specified time. A signal may be a command or a predefined message. A 'post' is the originating point of the form instance(s) or the signal. It could be eithera work station, a user, or a form process. Triggering can be graphically represented as:

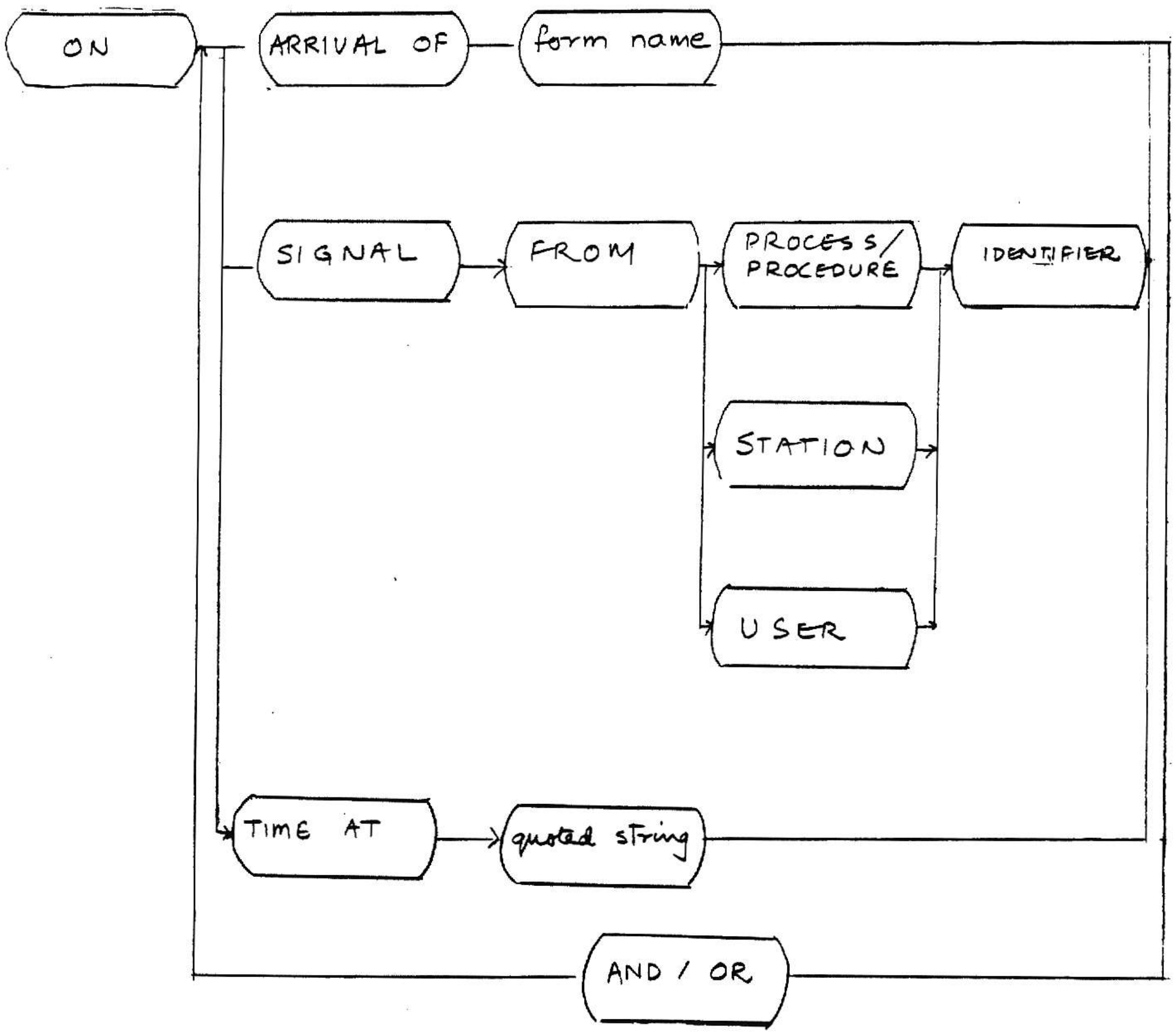

Fig.7 Trigger Specification

2.11 INVOCATION OF A FORM PROCESS

Invocation of a form process is simply expressed by a PERFORM statement, the syntax of which is : PERFORM <stmt>. Naming several processes in a single PERFORM statement implies that the named processes can be executed in parallel.

2.11.1 ROUTING (16, 17)

Routing refers to the electronic distribution of messages or forms. This can be specified in a routing syntax diagram as shown:

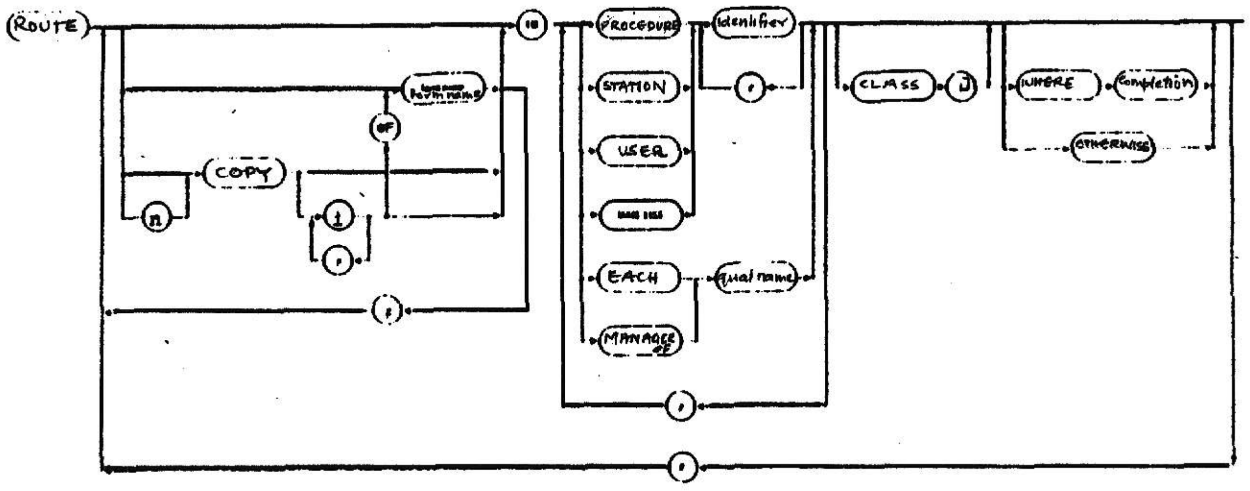

Fig.8 Routing Specification

A form name specifies what is to be routed, to which destinations, under what conditions. Also to provide for different priorities and services for different classes of mail, mail CLASS can be specified.

2.12 PROCEDURE DEFINITION

The procedure definition defines how to use trigger, perform, and route specifications to define a business procedure. This consists of a heading, followed by specification for a source of activity, and an END statement.

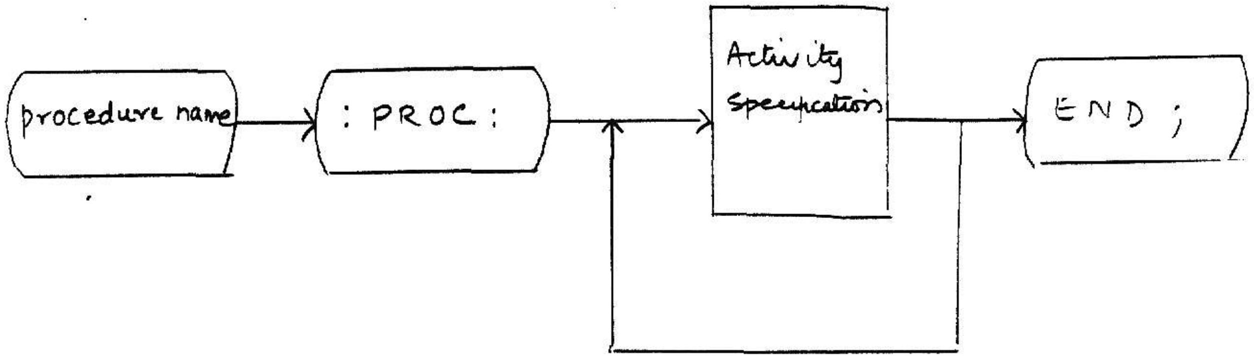

Fig.9. Procedure Definition

2.13 CONVERT: A HIGH LEVEL TRANSLATION DEFINITION LANGUAGE FOR DATA CONVERSION (15)

In the above named paper Shu, Housel and Lum of the IBM Research Laboratory in San Jose identified and researched an area which has direct bearing on the main point of this paper which is the development and use of forms as a design specification for network utilities which allow access through a common query facility to data at any node of a decentralized network of databases with non-homogenous database management systems (DBMS) query facilities.

Since in such a heterogenous environment we are dealing with different databases at different nodes, for effective communication between the nodes, data conversion must take place to a common medium which is recognizable to all nodes.

Their research describes just such a conversion language- they call it a translation definition language CONVERT, which provides very powerful and highly flexible data restructuring capabilities. Its design is based on the simple underlying concept of a form which enables the users to visualize the translation process and thus makes data translation a simpler task.

In their work they have said that essentially two tools are required to execute the data conversion process: 1) a data definition language to describe the source and target data structures and 2) a translation definition language to specify the mapping of instances from a set of source files to a different set of target files.

The data definition language that they have defined is called DEFINE. This language is capable of describing most linearized data structures; it uses the input/output format in the conversion model.

Further in addition to reasons stated above, the following are external factors requiring conversion of data:

1) changing to a different hardware environment

2) conversion from a conventional file system to a database system

3) conversion from one database system to another

4) a change in application requirements

Some conversions can be fairly simple, others quite complex.

Shu, Housel and Lum placed emphasis on designing a data definition language for describing the logical and physical aspects of data for a wide variety of data collections that could be found at any node of a network. Statements in this language can then be used as a driver for data translation. Their emphasis on the need for such a translation definition language has been recognized. Data conversion involving extensive and selective restructuring is becoming more and more common. CONVERT, provides powerful and flexible restructuring capability. They have assumed that users of CONVERT know the logical aspects of their data; know what they want to be done but do not want to be burdened with the details of how to accomplish it. CONVERT is a high level, easy to learn, procedural language for that kind of user. It handles all kinds of data structures with equal facility.

Since hierarchical data is the most abundant form of existing data it plays a dominant role in CONVERT's development. They envisioned that a translation analyst can best view his data in terms of Forms. According to their approach a Form is a two dimensional representation of hierarchical data which reflects the images of data instances. They have gone on to describe the hierarchical base of a form and the links between headings, subheadings and instances in terms of ancestors (parents) and descendents (children) as has been described earlier in this paper.

They have recognized two broad categories of translation definition in CONVERT: data mapping and data validation. Since the primary purpose of data conversion is to construct target data from various COMPONENTS of source data the primary emphasis is on data mapping. In addition recognition of invalid data is a neccessary part of the process.

The data mapping and restructuring facilities in CONVERT are provided by a set of form operators which include component extraction, SELECT, SLICE, GRAFT, CONCAT, MERGE, SORT, ELIM-DUP, CONSOLIDATE, a set of built in functions (SUM, MAX, MIN, AVG, and COUNT), assignment, and CASE- assignment. Each of these form operators operates on one or more forms (or their components) and produces a FORM as a result. The resultant FORM can then be used as an operand for another FORM extraction. All FORM operations can be nested.

The authors describe the notation that they have used in their language: F denotes a FORM; f denotes a field; C denotes a component of a FORM; EXPR denotes an arithmetic expression derivable from the fields of a FORM. An EXPR could be a constant, a field name or a built in function (e.g. SUM, MAX, MIN, AVG, COUNT); an expression derived from the above or a derived expression enclosed in parentheses using the +, -, *, / as arithmetic operators; or a sub-FORM. Specified conditions (SC) can define logical factors connected by AND's and/or OR's. The permissible comparison operators include =, #, <, >, \<, \>, >=, <=. Boolean values can be assigned and unless parentheses specify the priorities of evaluation, the logical factors are evaluated in standard left to right order.

Some of the form operations are:

2.13.1 FORM OPERATIONS (15)

1) ASSIGNMENT

Assignment takes the result of the operation(s) specified on the RHS of the assign operator (<--) and assigns it to the form named on the LHS.

Ex. 11 <--- POR (P#, S#, QR) produces a FORM 11 with column headings P#, S#, QR.

2) SELECT([EXPR1,....,EXPRN] FROM F [,...] [;SC].

This operation selects part(s) of a FORM if the specified conditions are specified.

Thus if a certain PERSONNEL form were as shown in Fig.10.

		EDUCATION				KIDS	
E#	NAME	SCHOOL	DEG	FIELDS	SAL	KNAME	AGE
1	JONES	A	-	CS	10K	MARY	10
		B	B	CS		JACK	8
						SUE	5
2	SMITH	A	B	BIO	20K	JACK	7
		C	M	CHEM			
			P	BIOCHEM			
3	DOW	A	B	MATH	15K		
				CS			
4	CARY	D	B	CHEM	18K	MARY	6
		B	M	CHEM			
5	JONES	C	B	MATH	25K	JILL	11
		B	B	PHYSICS		SUE	5
		D	P	MATH		JOHN	3
				PHYSICS			

Fig.10 Sample Personnel Form

then the result of SELECT (FROM PERSONNEL : DEG = 'P') would be shown in Fig.11.

		EDUCATION				KIDS	
E#	NAME	SCHOOL	DEG	FIELDS	SAL	KNAME	AGE
2	SMITH	C	P	BIOCHEM	20K	JACK	7

Fig.11 Result of SELECT from Fig.10

3. SLICE (f1,.....,fj FROM F)

The SLICE operation provides the capability to produce one row for each instance of fj

Ex. The result of SLICE (E#, DEG, FIELDS FROM PERSONNEL) would be as shown in Fig.12.

E#	DEG	FIELDS
1	-	CS
1	B	CS
2	B	BIO
2	M	CHEM
2	P	BIOCHEM
.		
.		

Fig.12 Result of SLICE from Fig.10

Thus, the SLICE operation provides a convenient means to produce relational tables from hierarchical structures. Each SLICE operation produces only one relational table.

```
4. SORT ( F [BY [ ASCENDING ] f, f2,.....,fN] [ WITHIN PARENT])
                  DESCENDING
```

The SORT operation sorts the instances of a FORM in either ascending or descending order of f1, f2,......,fn where f1, f2,.....,fn are members in the same part of a tree. The sort fields should be listed from left to right in order of decreasing significance, regardless of whether they are ascending or descending. If the WITHIN PARENT clause is specified, sorting will be performed over instances of the sort fields without affecting the sequences of the parent instances

Ex. SORT (F4 BY P# WITHIN PARENT) produces F4A as shown in Fig.13.

F4

S3	P#
A	10
	11
	12
C	15
	14
D	10
	12
B	13
	10

F4A

S#	P#
A	10
	11
	12
C	14
	15
D	10
	12
B	10
	13

Fig.13 Result (F4A) of SORT from F4

5. CONSOLIDATE (F FOR UNIQUE { f1, f2, ...
{(f1, f2,....), (fa, fb,...)...})

Duplicate instances in a field are removed as shown in Fig.14

Ex.

S#	P#	QTY
S1	P1	3
S1	P2	2
S1	P3	4
S1	P4	2
S1	P5	1
S1	P6	1
S2	P1	3
S2	P2	4

S#	P#	QTY
S1	P1	3
	P2	2
	P3	4
	P4	2
	P5	1
	P6	1
S2	P1	3
	P2	4

Fig.14 Result of CONSOLIDATE

There are other versions of the CONSOLIDATE operation.

6. GRAFT (F1, F2,......ONTO Fn [AT f] [: SC])

Graft provides a means to combine two or more FORMS into one FORM when specified conditions are specified.

Ex. Suppose we wish to form one file from the PTS and INV files such that the resulting file will have the information of the PTS file plus the quantity on hand (QH) obtained from INV which can be stated as shown in Fig.15.

GRAFT (INV ONTO PTS: PTS. P# = INV.P#);

PTS

P#	DES	S#	S CN	S UC
2	X	4	AB	5
		2	AB	4
3	XX	4	AB	2
		1	XB	3
7	Y	7	C	7

INV

P#	QH
2	10
3	17
4	5
7	20

P#	DES	S#	S CN	S UC	QH
2	X	4	AB	5	10
		2	BB	4	
3	XX	4	AB	2	17
		1	XB	3	
7	Y	7	C	7	20

Fig.15 Result of GRAFT

A PREVAIL field may be used in the GRAFT operation. The names on the LHS of the key word PREVAIL are considered to be the prevailing fields.

7.BUILT-IN FUNCTIONS

```
{ SUM }
{ MAX }
{ MIN }       ( f IN F [FOR UNIQUE f1,....,fn][:SC])
{ AVG }
```

{ COUNT }

The built in functions compute the sum, maximum, minimum, average, or count of the instances of a certain field f in a form F where the specified conditions are satisfied. They all have exactly the same format and operate in exactly the same manner. An example is shown in Fig.16.

Given

F

A	B	C	D	E
Q	1	2	3	4
		15	6	
		7	8	
R	2	9	10	11
		22	13	
S	3	14	15	16
T	1	17	18	19
		20	21	
		32	23	
	2	24	25	26

Fig.16 Form F

Then for SUM (C IN F) result in

2 + 15 + 7 + 9 + 22 + 14 + 17 + 20 + 32 + 24

the COUNT (C IN F : C<D) result is 7.

8. CASE Assignment

Every one of the form operators discussed so far performs one uniform operation over all instances of the relevant form(s). CASE Assignment, on the other hand, allows varied operations to be performed over different instances. These varied instances must produce homogenous results to be assigned to the resulting form for example:

```
F  <---  CASE ( f COP vi, v2, ...., vn [, others)]
         (F1, F2,...,Fn [,Fn + 1]);
```

F and f denote a Form and a field respectively and COP denotes a comparison operator, vi denotes a single value defined as:

```
<Single- VAlue> ::= <Value>| <Single-Value> OR <Value>
<Value> ::= <Literal> ANY OF <FORM>
```

and <FORM>, in turn, is either a Form name or a nestable Form operation representing a one-column Form.

2.14 EXPRESSING DIFFERENT DATA STRUCTURES IN TERMS OF FORMS (15)

Let us visualise a network as Fig. 17

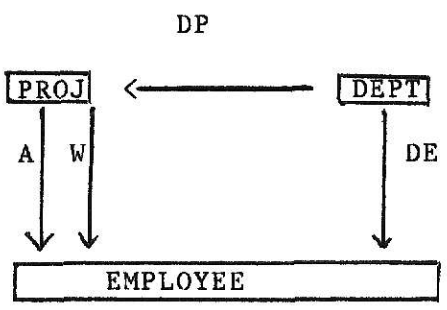

Fig. 17 Network of Forms

where each node in the network can be viewed as a form. For example the nodes DEPT and EMPLOYEE are shown as Forms in Fig.18.

DEPT

D#	MGR	P#	E#
55	SMITH	P1	551
		P2	552
		P3	553
			554
			555
			556
54	JONES	P1	541
		P4	542
			543
			544

EMPLOYEE

	EDUCATION		
EMP#	DEG	YR	SKILL
-	-	-	-
	-	-	
-	-	-	-

Fig.18 Nodes Dept. and Employee shown as Forms

Each named edge represents a means of connecting two forms. Conceptually, there are two ways to provide these connections. One way is to have the connecting information embedded in one or both of the Forms. Another is to build a Form to represent the information expressed by the edges. In the example, the edges DE and DP are embedded in the DEPT form, while A and W appear as separate Forms as shown in Fig.19.

PROJ

P#	LEADER	BUDGET
P1	541	100K
P2	554	200K
P3		50K
P4	542	300K

A

P#	EMP#
P1	541
P2	551
P4	541

W

P#	EMP#
P1	551
	552
	542
P2	553
	554
	555
P3	555
	556
P4	542
	543
	544

Fig.19 Resultant Forms Representing Information expressed by the edges

The user may decide to express the edge that serves as a connection between 2 forms. Conceptually, as Lum, Housel and Shu have described it, it is possible to adopt the notion of a Form as a basis for more complex data structures.

Thus a language can be specified to map instances of source items, which may be components of one or more files into instances of target data which may constitute multiple files.

It was found that if CONVERT was used the conversion could be performed using five to twenty times fewer statements than using a language like PL/1. Thus, through the medium of FORMS and a data translation language like CONVERT data conversion at nodes can be performed without problems.

Chapter 3

FORMS PROGRAMMING - OTHER VIEWS

3.1 DAVID W. EMBLEY (4, 5, 6, 7)

David Embley, of Dept. of Computer Science at the Brigham Young University has done extensive research on forms programming systems. Three of his papers are briefly discussed here.

In his paper "Forms Based Automatic Program Generation" he has described an approach to automatic program generation based on descriptions of data processing operations by means of conventional administrative forms. In order for forms based programming to proceed smoothly, a library of knowledge about common items of data needs to be available. The representation of this knowledge can be manifest in what the author calls a data frame. In a data frame, the set of data objects is described by giving:

1) the internal representation,

2) a routine to validate set membership,

3) a routine to transform a sequence of characters to the internal

representation,

4) a routine to transform the internal representation to a sequence of characters,

5) an initialization routine if needed.

According to Embley a form F is a 3 - tuple

$$F = \{ I, R, C \}$$

where

I is a nonempty, finite set of items

R is a set of relationships among the items

C is a set of symbols (or characters) that appear on the form

With a library of data frames, knowledge about forms and program templates available, an automatic program generator can work interactively with an application "programmer" to produce a desired object program

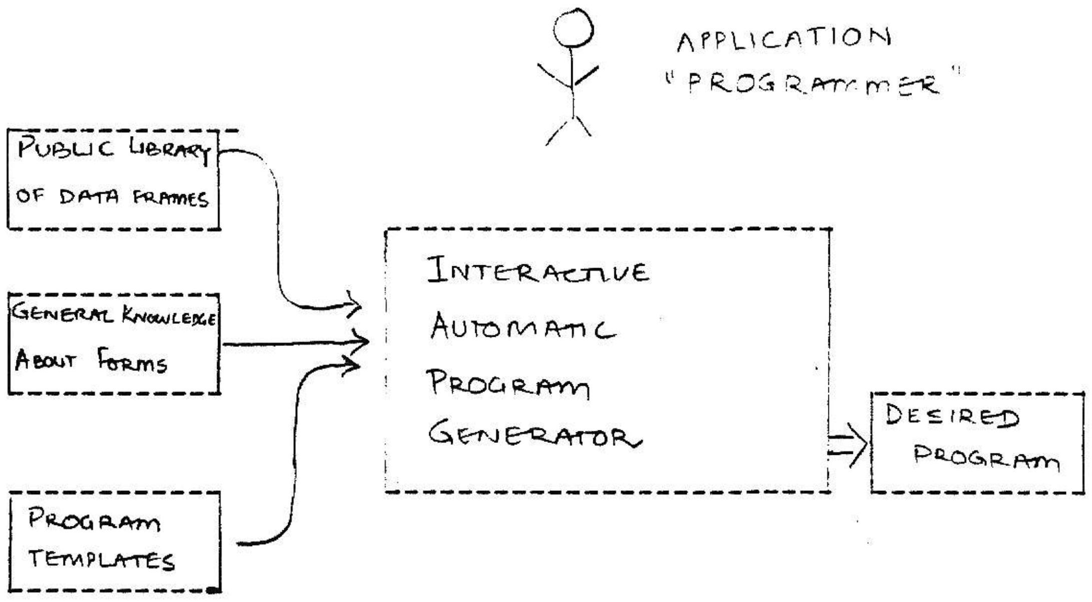
APPLICATION "PROGRAMMER"
PUBLIC LIBRARY OF DATA FRAMES
GENERAL KNOWLEDGE ABOUT FORMS
PROGRAM TEMPLATES
INTERACTIVE AUTOMATIC PROGRAM GENERATOR
DESIRED PROGRAM

Fig.20 Schematic of Automatic Program Generator

The application "programmer" need never see the object program and can, in fact, develop or modify it only through the program generator. The analysis phase of program development is still essential, but the actual coding is replaced by the design of the forms.

The dialogue proceeds as the programmer is prompted to supply the item I for a form and the relationships R among the items; the set of characters C is a by product of the process. For each item, the system requests a key phrase Ki, the type of blank space Bi, and the name of the data frame, which defines the possible entries Ei. The interactive program generator, in essence, is a data entry system that solicits extremely specific information from the programmer.

The public library of data frames is available for general use.

but only the data frames needed for the entries Ei become part of the object program. In addition to using a data frame available in the library, a programmer can also define application- specific data frames or take data frames from the library and modify them.

There are two categories of program templates, namely data entry and relationship validation templates. The data entry templates correspond directly to the five types of blank spaces. The "n entries" type of blank, for example, corresponds to a template that successively requests n entries and checks each one to determine data type compatibility. The relationship validation program templates make use of both blank types and predicates and functions defined in the data frames being used. To validate a total cost entry, for instance, the template makes use of the addition operator and the equality predicate in a dollar amount data frame and provides a summing loop based on the type of blank specified by the programmer.

Although the automatic programming process is presented here as being interactive, there are other alternatives. All that is essential is to supply for each item a key phrase Ki, a blank space type Bi, and a data frame name to define the possible entries Ei and to supply the relationships. Any method that communicates this information in a reasonable fashion is applicable.

Thus, with a library of data frames available, it is conceivable that a forms based programming system could select the appropriate frame for each item on the form and define relationships among the items without any intervention from an applications programmer.

In another paper entitled " A Forms Programming System" Embley proposes a forms based programming system (FPS) and discusses current efforts and expectations. The major objective of the FPS project is to investigate the extent to which ordinary forms can be used to provide a basis for a very high level programming system. Questions are explored such as what information can be inferred from an ordinary form about the data that is to flow to and from the form, and what additional information must be specified in order to generate software to direct the flow of this information. In partial answer to these questions, FPS provides facilities to analyze forms including the ability to match entry blanks with domains of expected entries, to generate queries for database retrieval and update, and to allow for the specification of the flow of information among forms. In applications where forms are common, it appears that an FPS system in which an application programmer specifies computer processes, directs information flows, and formats data by means of forms has promise of success.

3.2 D. TSICHRITZIS (18, 19)

OFS: An Integrated Form Management System

Tsichritzis has done a lot of work in this area. According to him there are three goals of office work - integration, evolutionary facilities and potential for automation. He describes forms as being electronic images of business paper forms. By handling forms there is a natural way to incorporate limited text capabilities together with ways to structure data for further retrieval and processing. He also describes form types and form instances. He describes three kinds of form fields - those that were created at form creation time and cannot be changed, those that will be created later and cannot be changed, and those whose value can be updated. Each station through which a form passes has a unique signature. Each time a field value is entered or changed on a form the system retains the system signature that initiated the change. With additional information about the time of the change plus the identity of the person connected to the station during that time a complete accountability of actions is possible on the form values i.e. each value can be uniquely attributed to a station and eventually to a person operating from that station.
station.

Each form instance has a unique key which identifies that form instance. A form can never be destroyed or deleted from the system after it is created. It can only be disposed of by sending it to a special disposal station. Such stations retain the privelege to shred electronically or archive the unwanted form. A form cannot be copied freely. The operations on electronic images of forms mirror operations on paper forms. Form instances can be filled and entered by displaying the appropriate format and allowing the user to enter values using a limited text editor to correct mistakes. Forms are stored in form files. Stations can only operate on forms in their local environment. A form has to be moved into a station before it can be retrieved, changed, etc., by that station. Forms move from station to station using mail commands. Forms can be located and traced in the system. All form commands are issued from a terminal via a short dialog with the system using reasonably friendly simple language. Forms can be printed if there is a need for paper output.

Tsichritzis talks of three kinds of office procedures. The first type called a desk activity, specifies a specfic action in a local station which is initiated under specific conditions local to the station and notifies other objects local to the station. The second type is called mail activity which automatically routes mail i.e. when a form arrives at a specific mail tray the procedure is initiated and routes the form to another mail tray according to the form's origin, contents, and the status of the system as it can be detected by a general program. The third type, called a coordination activity, is initiated when a complementary set of forms arrive or are present in specific files or mail trays. When this happens, the coordination activity will initiate a procedure and notify one or more stations about the event. Desk activities are specified by filling a form. A mail activity encapsulates automatic routing of messages. Mail activities are specified by filling an appropriate form. It distributes mail it receives according to predefined specifications. Mail can be sent according to three conditions. First, the contents of the form may determine the routing, second the origin of the form may determine its routing and finally, a general condition may be specified to a program which checks for that condition and routes the form accordingly. Mail can be forwarded to specified stations or be distributed according to certain percentages to distribute the load to stations. A coordination activity helps to streamline the activities happening in different stations.

3.3 LADD & TSCIHRITZIS (12)

AN OFFICE FORM MODEL

In this paper Ladd and Tsichritzis say that the effectiveness of automated offices depends largely on the success of formally describing and analyzing the well defined portions of traditional offices. The need for formal descriptive and analytic tools gives rise to the study of formal models of offices. The form flow model described in this paper regards an office as a network of stations through which forms flow. Forms originate in some initial stations of the network, flow from station to station where they are processed, and terminate in some final stations. The model yields to many types of analyses. The graph theoretic, commodity flow and query network approaches are descibed in this paper.

Form flow for a simplified loan office is described as consisting of five stations: a receptionist (R), a processing clerk (PC), and manager (PM), and a credit clerk (CC) and manager (CM).

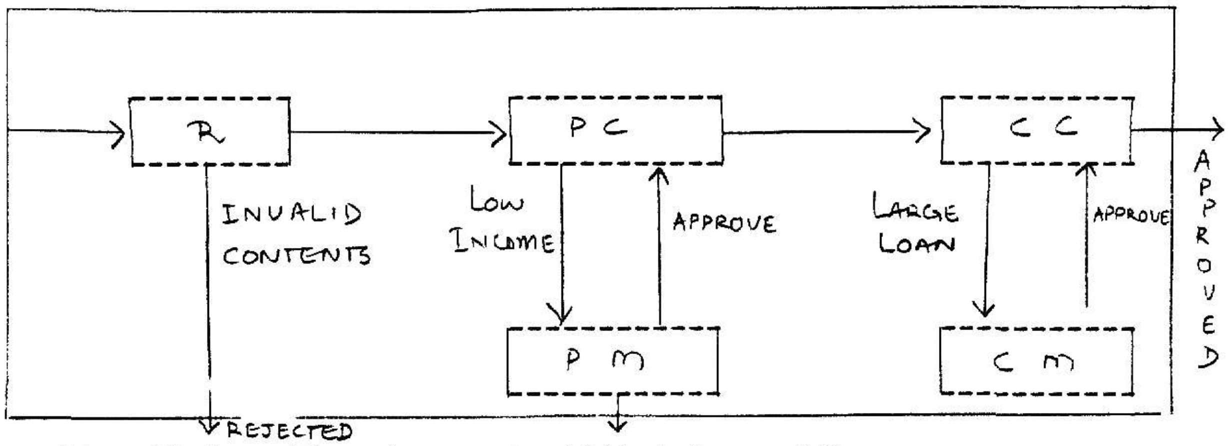

Fig. 21 Form Flow for a simplified loan office

R is given all the loan application forms. It considers those with invalid contents to be rejected overall and sends the rest to PC. PC first sends form low income applicant to PM for approval. Then PC sends the remaining forms and the forms returned by PM to CC. CC first sends those of large loans to CM for approval. Then CC considers the remaining and the forms returned by CM to be approved overall. PM and CM either approve or reject the forms they receive. They return the approved forms to PC and CC respectively and they consider the rejected forms to be rejected overall.

Each form contains fields for holding data values. An individual form (instance) carries three components (<St, Sk, Sc>): it's form type (St - the kind of fields it contains), its form key (Sk - a permanent unique identifier) and its form contents (SC - the field values).

Each station (Si) in the form flow model has a set of in-trays (Tii) and a set of out-trays (Toi) where forms are deposited. The task of a station is to take a form (<St, Sk, Sc>) from an in-tray (x (Tii), apply an operation selection function associated with the in-tray (Rx : {<St, Sk, Sc>} ---> Toi) to determine to which out-tray (Y (range (Rx)) the form is to be transferred, perform an operation associated with the in-tray-out-tray pair (@xy : { <St, Sk, Sc>}----> {<St, Sk, Sc'>}) and deposit the transferred form (<St, Sk, Sc'>) in that out-tray (Y). A *flow arc* ((X,Y) (E i) exists between the in-tray (X) and an out-tray (Y) if it is potentially possible for the operation selection function associated with that in-tray to select that out-tray (y(range (Rx)). Each operation flow arc represents an operation, depending upon the analysis, costs (times, weights, capabilities, etc.) may be associated with the flow arcs (Ea = U Eai). It may be desirable to abstract out operation selection functions by estimating the frequencies of selecting the different flow arcs in the network. The network also designates a subset of all trays (To = UiToi) as final trays (Tw (To) where forms terminate in the network. The task of the network is to take a form (<St, Sk, Sc>) from a non final out tray (Y (To - Tw), apply a routing selection function associated with the out-tray (Ty : {<St, Sk, Sc>} --->Ti) to determine which in-tray (X (range (Ry)) the form is to be transferredto, and deposit the form in that in-tray (X). A flow arc ((Y,X) (Ec) exists between a non-final out-tray (Y) and an in-tray(X) if it is potentially possible for the routing selection function associated with that out-tray to select that in-tray (X (range (Ry)). Since each routing flow arc represents a communication link between the two stations, depending on the analysis, costs may be associated with the flow arcs.

Forms are conserved in the network. They are neither created nor destroyed, only transferred.

Tsichritzis has also worked on optimal paths of different forms and the time taken to process them and their frequency. He has done these in graph-theoretic analysis. Through commodity flow analysis, a form flow network can be augmented to analyze its capacity in terms of maximum flow. The problem here is to maximize subject to constraints,over the capacities of the stations. Each station is assigned a value indicating the maximum number of work units it can perform per unit of time. Linear programming can be used to solve the network problem amidst constraints. He has also performed queing network analysis on form networks where each form becomes a job. Each station becomes a server with a single queu corresponding to its set of in-trays. Each operation in a station is associated with a different job class. The routing frequencies which correspond to the frequencies of taking the routing flow arcs must be given. The queing networks can be solved exactly by analytic techniques provided some further assumptions and restrictions are made.

The scheduling disciplines are restricted to be FCFS (first come first served), PS (processor sharing simultaneously among all jobs in the queu), NQ (no queuing - the server supplies as many processors running at the full rate as there are jobs in the queu), or PLCFS (pre-emptive last come first served). If the scheduling discipline for a server is FCFS, then all service time distributions for the server are restricted to be identically exponential.

3.4 OTHER RESEARCH - TSICHRITZIS (18, 19)

In another paper Tsichritzis writes about a small editor associated with filling entries on forms to allow correction of mistakes. The system guides the user to fill all the neccessary fields and retains the station number. In this prototype system there is no form deletion operation. By controlling form instance deletion one can impose a law of conservation on forms. Forms originate in particular stations. Forms terminate only in disposal stations. There is no other way for a form to exit the system. Hence, at any point in time the system can find out exactly how many form instances there are and, using the locate commands determine where the forms are at any moment in the processing cycle. He talks about a very important set of commands which deal with moving forms between stations. These operations maintain an electronic mail facility which keeps a running account of where all the forms are for both original and official copies. These logs provide three functions in the system. First, they can be used to give the overall status of the system regarding bottlenecks in communication or overloading the of stations. Second, they can be used for locating forms in stations. Third, they can be used for recovery when a station malfunctions. All mailed forms in this system pass through a control node where a log is kept about the movement of the forms. This log enables the system to locate and trace forms. A trace command provides a complete path that a form took from its originating station up to where it is at the moment.

Forms also should be available for database operations. All form instances in this implementation can be accessed using relational commands. The operations are similar to SQL and are provided by a relational database system called MRS. A command can be as simple as:

```
SELECT Customer name FROM Invoice WHERE Status = payable
```

Database operations are in the format of such a relational query language. Hence, database operations alongwith form operations are easily realized in terms of forms. There is a facility for the system to handle distributed queries whereby a global query can be issued from a station and this in turn translates to a series of local queries on local stations.

The specification of an automatic procedure in this prototype system is provided by the TLA (Toronto Latest Acronym). Every form manipulated by a form procedure usually has a precondition sketch and an action sketch. A form 'precondition sketch' indicates a request to the system to find 'a form that looks like this'. An 'action sketch' indicates a request to modify a form that has already been obtained. Restrictions are placed on the values of form instances, and the which of which causes automatic procedures to be called.

The working set of form procedures is abstracted in terms of a sketch graph. The authors talk of other models for form flow operations such as flow analysis and station restructuring, the mathematical details of which are not pertinent to the scope of this paper.

2.14.5 KITAGAWA AND KUNII

In their paper 'Form Transformer' Kitagawa and Kunii have proposed a new model called the Nested Table Data Model (NTD). In this model they call forms nested tables and they have given different names to the fields that occur on the form. They have described four Form Transformer operators: Column Ennest (CE) operator; Column Denest (CD) operator; Row Ennest (RE) operator and Row Denest (RD) operator. The first two operators handle column nesting; the `ennest' column operators nest columns or rows, and the `denest' operators reduce the nests. They are also called form transformer operators and some of them are included as other names in the language CONVERT described earlier in this paper. The authors have shown some propositions and corollaries on the basic properties of forms, placing emphasis on the assurance of reversability and commutability of form transformation. They also clarified that, in certain conditions, nested tables can be supported as user views to flat tables without tedious view update problems. The authors are currently developing a prototype form handling system based on NTD especially aiming at software management office automation.

Chapter 4

CONCLUSION

This paper has basically surveyed different aspects of forms processing from papers written on this subject as late as 1983. The methodology of specification was discussed in detail and data and process descriptions were researched. Several form operations applied to forms giving results as another set of forms were highlighted. The paper also surveyed a procedure definition language and how form operations could be triggered or routed. Since forms may operate in a heterogenous distributed environment a high level Translation Definition Language for Data Conversion neccessary at different nodes of the network was described. This translation definition language is CONVERT with powerful data restructuring capabilities. The mapping between different nodes with heterogenous systems is made simpler through the medium of forms which permit a common interface between nodes, since a form is the same for different systems and for different nodes. The data translation is much simpler and simple operations to produce resultant forms can be affected with relative ease making possible communication between two heterogenous nodes of a forms network.

Broadly, each instance in a form would be stored in an integrated directory/dictionary. But copying of parts of this integrated directory/dictionary at various nodes depending on frequency of accesses would depend on the system. Some guidelines as to when this would be done have been discussed in the paper. Embley has discussed these form instances as data frames and has said if there is a directory of data frames then forms processing becomes simple. He has also described the schematics of automatic program generation. These programs are automatically generated upon invocation by form processes. These invocations are done by an operating system written specially to handle them. Thus, with a library of data frames at its disposal, it is conceivable that a forms based programming system could select the appropriate frame for each item on the form and define relationships among items without any intervention from the application programmer.

Some office flow models describing the business environment as a network of stations through which forms flow were discussed. In particular, the pioneering work of D. Tsichritzis was highlighted especially his work on the optimal flow paths of differnt forms in a network and the time taken to process them based upon their frequency. His graph theoretic, commodity flow and queuing network analysis as related to forms were briefly described in the paper.

As such this paper is an up-to-date research on work done on forms programming systems and incorporates all major publications on this paper to date.

It is possible to show, that a good solution to heterogeneity in a distributed database system is to make it transparent from an operational viewpoint, by providing each local database with a standard interface i.e. forms, for distributed operation.

Indeed, forms allow access through a common query facility to data at any node of a decentralized network of databases with non-homogenous data models, non-homogenous data model implementations and non-homogenous database management systems.

The above survey lends credence to a proposal, put forward by many authorities on heterogenous database management systems, of having as a query language, a language built around documents or forms. Clearly, in the business or data processing environment, documents serve as an all purpose media for communicating information. A document environment with spread sheet capabilities and which uses either global or local names provides a readily acceptable format, a transparent mechanism without regard to existing database structures, and a representation which is translatable or mappable into any number of database models. The global and local names will be handled as specified by the data dictionary system (DICS) / directory system (DIRS). This also minimizes the redundancy of data duplication at several nodes of a distributed network.

Using this standard mechanism of documents as a query language, it is relatively easy to provide a mapping from the document itself to an actual query. One can derive programs to satisfy information requests from the document format. In some cases, the mapping from the document may be a simple request in a standard query language; in other cases where the data is distributed over several systems or models, the request may be mapped into a more complex form such as programs.

BIBLIOGRAPHY

1. Chen, P. S., The entity-relationship model: toward a unified view of data, ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976.

2. deJong, S. P., The system for business automation (SBA): a unified application development system, Information Processing 80,Tokyo, Japan, and Melbourne, Australia, October 1980, pp. 469-474.

3. deJong, S. P., and Byrd, Roy J., Intelligent forms creation in the system for business automation (SBA), Research Report RC8529, IBM T. J. Watson Research Center, Yorktown Heights, New York, October 1980.

4. Embley, D. W., A forms-based nonprocedural programming system, Technical Report, Department of Computer Science, University of Nebraska, Lincoln, Nebrasks, October 1980a.

5. Embley, D. W., A forms-based programming system, to appear in Advances in Database Systems Management, Vol II, P. Fisher, J. Slonim, and E. A. Unger (eds.), Heyden & Sons, 1984.

6. Embley, D. W., Forms-based automatic programming, ACM 78 Proceedings, Washington, D.C., December 1978, pp. 972-979.

7. Embley, D. W., Programming with data frames for everyday data items, NCC 80 Proceedings, Vol 49, Anaheim, California, May 1980b pp. 301-305.

8. Gehaui, N. H., "The Potential of Forms in Office Automation," IEEE Transactions, Vol. Com-30, No. 1, Jan 1982.

9. Hogg, J., Nierstrasz, O. M., and Tsichritzis, D.C., Form procedures, in Omega alpha, Tsichritzis (ed.), Technical Report CSRT-127, Computer Systems Research Group, University of Toronto, Toronto, Canada, March 1981, pp. 103-133.

10. Kitagawa H., Kunii T. L., "Form Transformer - Formal Aspects of Table Nests Manipulation," Department of Information Science, University of Tokoyo, Japan, 1980.

11. Kitagawa, H., Kunii, T. L., Harada, M., Kaihara, S., and Ohbo, N., A language for office form processing (OFP) - with application to medical forms, Department of Information Science, University of Tokyo, Tokyo, Japan, 1980.

12. Ladd, I. and Tsichritzis, D.C., An office form flow model, NCC 80 Proceedings, Anaheim, California, May 1980, pp. 533-539.

13. Luo, D. and Yao, S. B., Form operation by example - a language for office information processing, Proc. International

Conference on Management of Data, Ann Arbor, Michigan, May 1981.

14. Lum, V. Y., Shu, N. C., Tung, F. C., and Chang, C. L. Automating business procedures with form processing, Research Report RJ3050, IBM Research Lab., San Jose, California, March 1981.

15. Shu, N. C., Hourel B. C., Lum V. Y., "CONVERT: A High Level Transaction Definition Language for Data Conversion," Communications of the ACM, Vol 18, No. 10, October 1975.

16. Shu, N. C., Lum, V. Y., Tung, F. C., and Chang, C. L., Specification of forms processing and business procedures for office automation, Research Report RJ3040, IBM Research Lab., San Jose, California, February 1981.

17. Shu N. C., Wong H. K. T., Lum V. Y., "Forms Approach to Application Specification for Database Design" IBM Research Report RJ2687 (34432), IBM Research Laboratory, San Jose, California, February 1980.

18. Tsichritzis, D. C., OFS: an integrated form management system, in A panache of DBMS ideas III. Tsichritzis, D. C. (ed.) Technical Report CSRG-111, Computer Systems Research Group, University of Toronto, Toronto, Canada, April 1980, pp. 1-17.

19. Tsichritzis, D. C., Form management, in Omega Alpha, Tsichritzis, D. C., (ed.), Technical Report CSRG-127, Computer Systems Research Group, University of Toronto, Toronto, Canada, March 1981, pp. 26-100.

20. Zloof, M. M., Query-by-example: a data base language, IBM Systems Journal, Vol. 16, No. 4, December 1977, pp. 323-343.

21. Zloof, M. M., A language for office and business automation, Research Report RC8091, Yorktown Heights, New York, January 1980.

A SURVEY OF FORMS PROCESSING TECHNIQUES

By

RAJIV KAPOOR

MBA, KANSAS STATE UNIVERSITY, 1982

ABSTRACT OF A MASTER'S REPORT

Submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

Department of Computer Science

KANSAS STATE UNIVERSITY

Manhattan, Kansas

- 1984 -

ABSTRACT

The objective of this research is concerned with the development and use of forms as a design specification for network utilities which allow access through a common query facility to data at any node of a decentralized network of databases with non-homogenous data model implementations and non-homogenous database management systems (DBMS) query facilities.

The fact that the query language syntax varies widely is a major problem in a network of distributed databases. Some queries are meaningless to other data models; they only have relevance in models with a particular structure. To solve this problem, a normalized query facility/ user interface is required.

A language built around documents would resolve the above problem. Documents serve as an all-purpose media for communicating information. A document environment with spread sheet capabilities and which uses either global or local names provides a readily acceptable format, a transparent mechanism without regard to existing database structures, and a representation which is translatable or mappable into any number of database models. The global and local names will be handled as specified by the DD/DB facility.

Using a forms based query language, it is easy to provide a

mapping from the document to the actual query. One can derive programs to satisfy information requests from the document format. In some cases, where the data is distributed over several systems or models, the request may be mapped into a more complex form such as a program.