A System for Managing Structured Messages

DENNIS TSICHRITZIS, FAUSTO A. RABITTI, SIMON GIBBS, OSCAR NIERSTRASZ, AND JOHN HOGG

Abstract—**Message systems send and receive messages but do not manage the information the messages contain. Database management systems manage the information of a global database but do not have a notion of address. In this paper we outline a prototype system which integrates the facilities of message systems and database management systems. The system manages structured messages according to their contents. The messages can be stored within a station and transferred between stations. Information present in the messages can be queried in a distributed manner. Message structure can also be exploited by automatic procedures which recognize triggering conditions and perform user specified actions.**

I. INTRODUCTION

THE main purpose of message systems is to enable users to send and receive messages [7]. In current message systems, the messages remain uninterpreted (consisting of a *header* and *body*) [8]. The header incorporates structural data needed to identify and route the message. The body is a series of bytes which the message system is supposed to deliver intact. The main function of the system is to deliver each message to the right destination quickly and without error. The system is not expected to understand or care about what information the message contains. This situation, however, presents a major limitation on the flexibility and usefulness of the system. The system *delivers* the messages but does not *manage* the messages. Other systems are supposed to manage the information that the messages represent.

In order to enhance their functionality, message systems have to interpret, at least partially, the messages which they handle. In this way, users can query the message system to find messages, to accumulate data that the messages contain or to specify automatic processing and routing procedures which make use of the contents of messages. To interpret messages, the message systems need some guidance. Such guidance can be provided by superimposing some structure on the messages. This structure is known to the system and used for the interpretation of the message.

It can be argued that the separation of the header and its fields provides a certain structure which can be used for finding and routing messages. We claim, however, that this structure is not sufficient for two reasons. First, since the header structure is uniform for all messages, it cannot provide an adequate interpretation of all messages. Second, since the header structure is fixed it cannot evolve with time; this may prove to be inadequate as the system and its messages evolve. As an alternative to this fixed structure, we propose that messages be *structurally typed.* Thus, their structure can be declared and used to guide their interpretation. The partial interpretation of the messages enables the system to manipulate the message's contents, to find messages, and to route messages. In this way, message systems will manage the information that the messages contain.

We will outline an experimental message management system. The system enables users to enter messages in stations and send and receive messages through mailboxes. Users can utilize the contents of the messages to:

1) file and retrieve messages within their own stations;
2) locate messages in the message management system; and
3) query and obtain data present in messages distributed in a group of stations.

Users of our experimental system can also specify procedures which automatically operate upon messages on the basis of their contents. For instance, users can specify procedures which:

1) coordinate messages, i.e., act only when a related set of messages has been assembled;
2) modify and create messages;
3) file messages in dossiers; and
4) automatically forward received messages to other stations according to their contents.

The ability to manipulate messages and procedures is provided through a uniform user interface. This interface is based upon specification by example [18]. Users find and query messages by partially specifying the contents of the messages. The automatic procedures are also specified by giving the system some indication of the contents of the messages and what the system should do with the messages.

These facilities provide an added-value message system which treats messages according to their content and not as "sealed letters." This approach encourages the integration of message systems with office procedures performed on messages. For example, we can incorporate some of the functions of a secretary (who opens the mail) as part of the message system. In addition, systems with structurally typed messages allow the integration of message and database facilities [2]. Such versatility is one of the goals of office automation [5], [14].

II. SYSTEM OVERVIEW

The overall structure of the system is shown in Fig. 1. The system is composed of a number of logical units called *stations* which may be grouped together on physical units called *nodes.*

Manuscript received May 7, 1981; revised November 7, 1981.

D. Tsichritzis, S. Gibbs, O. Nierstrasz, and J. Hogg are with the Computer Systems Research Group, University of Toronto, Toronto, Ont., Canada.

F. A. Rabitti was with the Computer Systems Research Group, University of Toronto, Toronto, Ont., Canada. He is now with the Instituto Elaborazione Informazione, Pisa, Italy.

0090-6778/82/0100-0066$00.75

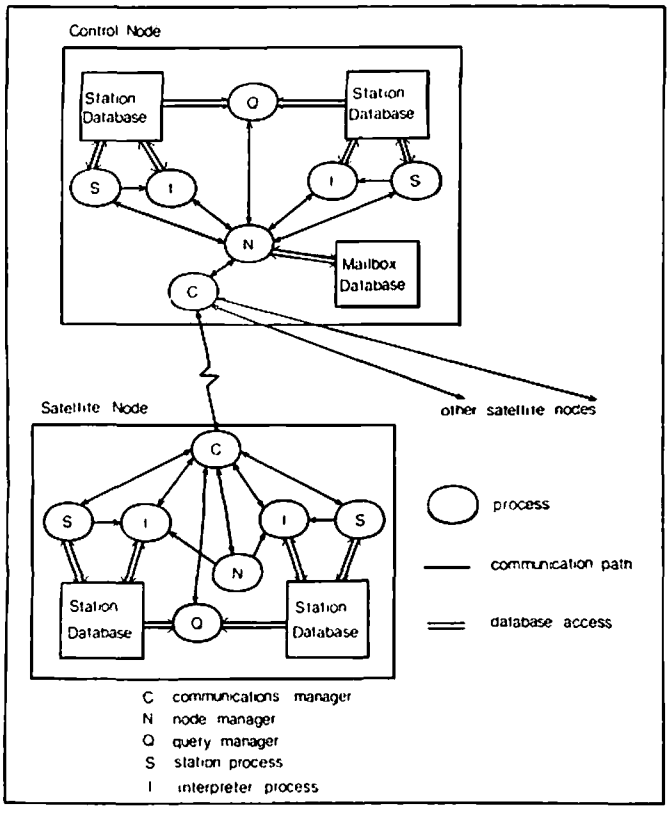

Fig. 1. System architecture.

A *control node* is chosen to perform synchronization and control activities; the remaining nodes are known as *satellite nodes.* A star-configured network is a natural choice for such an organization. In our prototype system each node is a PDP-11/23 processor. These are connected by a high bandwidth (6.4 Mbits/s) interprocessor communication link.

Each node supports a number of processes which are either associated with the node itself or a particular station on the node. The three per-node processes are as follows.

Communication manager acts as an intermediary when data must be transferred between processes located on separate nodes and hides the network from these processes.

Node manager is responsible for the activation of certain processes on the node. In addition, on the control node, the *control node manager* switches control information between processes and responds to a variety of service requests.

Query manager performs global queries on a single node.

The two per-station processes are as follows.

Station process accepts commands from the user and displays the results of performing these commands.

Interpreter process executes automatic procedures.

Each station contains a *Station Database* that is used to store information associated with the station. The MRS relational database management system [9] is used for all Station Databases. For each *message type* (such as a meeting announcement or a referee report) that is known by the system there is a relation in each Station Database. Tuples within such a relation correspond to instances of the message type. Attributes of this relation correspond to the fields of the message type.

III. MANUAL MESSAGE OPERATIONS

We assume that each user of the system operates a single station. Each station is characterized by a unique global identifier which distinguishes it from all other stations. Typically a station process is activated when a user logs on by supplying the station's identification and a password.

A user defines a message type by creating a display template. Once the type has been defined users may create message instances by "filling in" the template. During message instance creation the system generates a globally unique identifier for the new message instance. This identifier (known as the message key) is permanently attached to the message instance and cannot be modified. A message template and message instance are shown in Figs. 2 and 3.

Besides creating messages a user may also copy messages. If N copies are to be made of a particular message, then N new tuples are inserted into the Station Database. The new tuples are identical to the original except for their message keys. These are assigned in such a manner as to maintain global uniqueness.

Message creation and copying require inserting into the Station Database. It is also possible for a station to update tuples present in its Station Database. This operation is strictly controlled to prevent one station from modifying the contents of another's Station Database. Also there are constraints upon the modification of particular message fields. There are four types of message fields. The first three types are all user supplied. Fields of the first type must be entered during message creation and then cannot be modified. Fields of the second type cannot be modified once entered. Fields of the third type have no restrictions in terms of operations. The fourth type is used for *automatic* fields (such as dates, "signatures," message keys, and other functionally determined fields) which are generated by the system when required and cannot be modified by the user.

In order to modify or copy a previously created message it is first necessary to retrieve the message that is to be operated upon. Retrieval is performed either selectively by specifying a selection condition, or sequentially by message key. In neither case need the user be aware of physical access paths within the Station Database.

A station can mail messages from its Station Database to another station. The transfer is actually performed in two stages. During the first stage, known as the ship operation, a message is deleted from the Station Database of the issuing station and inserted into a special *Mailbox* database. The Mailboxes are located on the control node; this allows messages to be sent to a disconnected satellite. Tuples in a Mailbox are tagged with the identification of the destination station. The second stage of message transfer, known as the get operation, deletes the message from the Mailbox and inserts it into the Station Database of the issuing station. Thus a station must explicitly request its mail.

Whenever a ship or get operation is performed, an entry is appended to a log file. This file records the time of the operation as well as the source and destination stations and the key of the message being transferred. When the key of a message

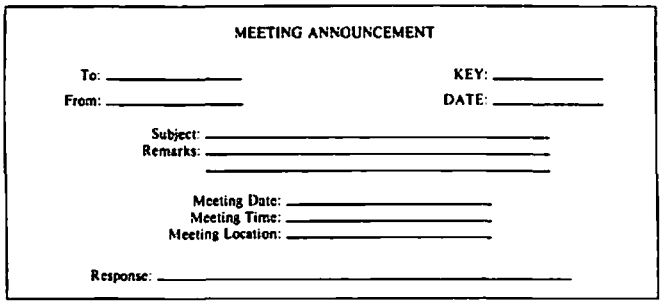

Fig. 2. A message template.

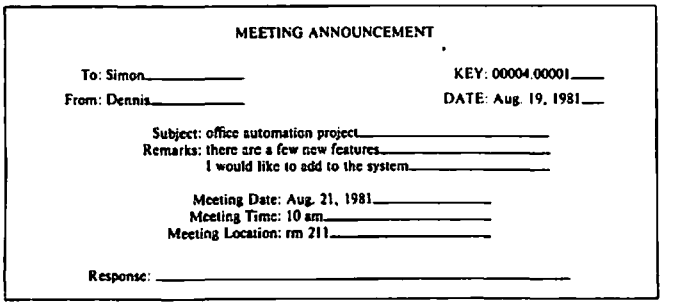

Fig. 3. A message instance.

is known it is possible to locate the message by examining the message's most recent log entry. Similarly, it is possible to produce a trace of a message, that is, a list of the stations it has visited. This is useful for the analysis of information flow within the office [10]. Both of these operations, locate and trace, can be issued by any station.

The Mailboxes, log file, and counters from which message keys are dispensed are located at the control node. Thus certain operations may be considered as requests for services from the control node. Requests issued from processes running on a satellite node are served by the node manager of the control node. (If the issuing process is on the control node these operations are much simpler since no communication over the network is involved.) As an example, consider message creation. In this case a station process sends a request for a new message key. The control node manager then obtains a key from one of its counters, increments the counter, and sends the key to the station process. As another example, during the ship operation the control node manager receives a message plus the identification of the destination station. The control node manager then inserts the message, now tagged with the destination, into a Mailbox and updates the log file.

Two major constraints on messages and their operations have been built into the system. First, all message instances are unique (both logically and physically). Second, a particular station can only operate on messages within its Station Database. The first constraint allows us to think of messages as objects, that is, they have a single location. The second constraint allows us to talk of message "ownership"—a message is owned (i.e., accessible) by the station in whose Station Database it resides. These two constraints force the system's electronic messages to behave like ordinary paper messages. In this way the system is in some sense "predictable," i.e., the operations of the system correspond to operations within the user's conceptual model of the office.

IV. MESSAGE QUERIES

Some office activities may require information which is spread in messages over more than one station. In such cases queries of information present in messages are useful. We will discuss how a station user specifies a query and how the result of a query is presented to the user. We will then present an outline of the implementation of the processing of queries.

As the first step in query specification the user selects the message type on which the query is to be performed. The message template is then displayed on the screen. A *query sketch* is then created by partially filling the template. This serves as an example and informs the system of the kinds of messages qualifying for the query. This approach has been used in FOBE, QBE, and OBE [11], [18], [19]. The user may fill in zero or more fields of the template; values entered into the fields are interpreted as selection conditions. For example, if the user enters ">10" into a field, then all messages which satisfy the query will have values greater than 10 in this field. In addition to ">" one can also specify "<", or "=", etc., as well as a pattern match. Such a condition is known as a simple condition. For each field it is possible to specify a field condition which is a disjunction of simple conditions. The messages that satisfy the query will satisfy the conjunction of all field conditions.

Once a query sketch has been created the user next specifies the scope of the query. The allowable choices are as follows.

Local: In this case the query is performed on the Station Database of the issuing station.

Group: In this case the query is performed on all Station Databases on the same node as the issuing station.

Global: In this case the query is performed on all databases in the network (this includes the Mailboxes at the control node).

Explicit: In this case the user lists the station names of the Station Databases that are to be searched.

After specifying the scope the query can be processed. The results are stored in a temporary database belonging to the issuing station. This database contains *message images* rather than messages; the messages themselves still reside in their respective Station Databases. A message image differs from a message in that it is temporary and read-only. It is also invisible to any of the automated procedures that process messages. The tuples from this temporary database may be displayed by the station. When this occurs the identification of the Station Database in which the message was found is also indicated.

Each query involves a single message type. It is not possible to directly specify operations involving more than one message type. However, arbitrarily complex multiple-type joins may be performed by first individually constructing complete temporary databases. The station user can then invoke a relational database system and express his query (now over the local temporary database) using a high-level set-oriented relational query language.

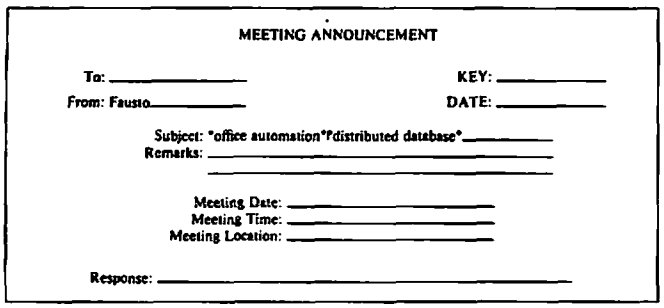

Fig. 4. A query sketch.

An example of a query sketch is shown in Fig. 4. This query will search for meeting announcements from "Fausto" and on the subject "office automation" or "distributed database." If this query is performed with a local scope then the meeting announcements sent to the station user (or at least residing at his station) will be searched. If this query is performed with a global scope then all meeting announcements in the system will be searched. By using the "Response" field of this message type it is also possible to determine who has replied to the meeting announcement.

V. QUERY PROCESSING

The strategy for processing a query is determined by the scope of the query. Again we distinguish the following cases.

Local: In this case the query manager is not used and the station process itself performs the query on its Station Database.

Group: In this case the station process sends the query to the query manager for the node. It then waits for an answer from the query manager.

Global/Explicit: In this case the station process sends the query to the control node manager. The control node manager then passes the query to all query managers within the scope of the query. This may include the query manager on the control node. The various query managers perform the query on their node and send the answer back to the control node manager. The control node manager assembles the answers in a temporary database and may also perform the query on the Mailboxes if they are included in the query's scope. Finally, the control node manager sends the temporary database to the station process which issued the query.

Two problems arise in the processing of queries: concurrency control of interfering global or local operations and control of data movement due to mailing operations. Two algorithms, the centralized concurrency control algorithm and the centralized movement control algorithm [13], are used to circumvent these problems. These algorithms are concerned with queries with a scope of more than one node.

The concurrency control problem [12] refers to the scheduling of operations which may conflict with queries. There are two such sources of interference, local updates and other queries. The system gives precedence to all local updates. Stations are allowed to modify, create, or copy messages even while queries are in progress. In addition, separate queries can operate concurrently. However, in this case, scheduling of the queries is required. For example, suppose data item X on station i initially has value a_1 and then is changed, by a local update operation, to value a_2. Similarly, on station j the data item Y is changed from b_1 to b_2. If we have two queries q_1 and q_2, it is possible that q_1 will see X as a_1 and Y as b_2 while q_2 will see X as a_2 and Y as b_1. Whether we consider q_1 as occurring before or after q_2 this result is inconsistent with the history of X and Y.

The source of this problem is that two distinct queries with overlapping scopes may be performed in different orders on different nodes. This problem can be solved by having the control node manager serialize query requests. Each query, when accepted by the control node manager, is given a progressive sequence number: *Seq*(query). This is similar to the use of timestamps [1]. However, since the *Seq* numbers are generated from a single node any sequential ordering can be used. Queries are sent by the control node manager to the satellite nodes in this order. The network protocol ensures that the order of queries sent from one node to another is equal to the order received. Since there is a single query manager at each node the queries are performed in this order, i.e., that of their *Seq* numbers.

The movement of messages from one Station Database to another (or for the Mailboxes) also introduces difficulties with query processing. In particular, the following pathological situations must be avoided.

The message M is missed by a query.

1) The query is performed on node i while the message M is on node j.

2) Message M is transferred to node i.

3) The query is performed on node j.

The message M is counted twice.

1) The query is performed on node i where it sees the message M.

2) Message M is transferred to node j.

3) The query is performed on node j where it again sees the message M.

A query is first performed on the Station Databases by the query managers and then on the Mailboxes by the control node manager. However, for messages that are transferred it is still necessary to keep track of the sequence number of the last query that has seen the message. For this purpose the following control relations are used to store the images of messages that have been transferred:

SEE (message, station, has-seen, will-see)
HIDE (message, has-seen)
$HIDE_i$ (message, has-seen)

The *SEE* relation is found on the control node and used for messages transferred from a Mailbox to a Station Database. Message images which appear in this relation will be found by the control node manager even though they reside at the station specified in the relation. The field *will-see* is the sequence number of the next global query to be performed on the node to which the message has been transferred.[1] *Has-seen* is the

[1] This interpretation of the control relations assumes that a particular message instance is transferred at most once during the processing of any query. In the more general case the interpretation is not as simple; however, the algorithms are unaffected [13].

sequence number of the most recent query that has seen the message (at any node). Any queries between *has-seen* and *will-see* must see the message at the control node.

The relations *HIDE* and $HIDE_i$ are similar except that the first is managed by the control node manager while $HIDE_i$ is managed by the query manager of the ith node. Thus there is a version of $HIDE_i$ on every node and a single *HIDE* on the control node. The field *has-seen* is the sequence number of the most recent query that has seen the message (at any node). Message images which appear in these relations will be overlooked even though the messages are present at the node.

Let q_i be the last query performed on node i and q_c the last query performed at the control node. The message ship and get operations are then

```
Procedure ship (message)     /*from node i*/

    At node i (performed by a station process):
        if the message is in HIDE_i then
            obtain has-seen from HIDE_i
            delete the message from HIDE_i
        else
            set has-seen equal to Seq(q_i)
        fi
        send the message and has-seen to the control node
        delete the message from the Station Database

    At the control node (performed by the node manager):
        insert the message into the Mailbox
        if has-seen is greater than Seq(q_c) then
            insert (message, has-seen) into HIDE
        fi

Procedure get (message)          /* to node i*/

    At the control node (performed by the node manager):
        if the message is in HIDE then
            obtain has-seen from HIDE
            delete it from HIDE
        else
            set has-seen equal to Seq(q_c)
        fi
        send the message and has-seen to node i
        receive Seq(q_i) from node i
        if Seq(q_i) is greater than has-seen then
            insert (message, i, has-seen, Seq(q_i) + 1)
                into SEE
        fi
        delete the message from the Mailbox

    At node i (performed by a station process):
        insert the message into the Station Database
        if has-seen is greater than Seq(q_i) then
            insert (message, has-seen) into HIDE_i
        fi
```

It is important to note that no query will be performed on the Mailboxes in the midst of a get or ship operation. This is because the single control node manager is responsible for these three operations. Moreover, the control node manager performs queries on the Mailboxes in the order of their *Seq* numbers. Assuming that the above conventions are followed by the get and ship operations, we may perform queries as follows:

```
Procedure query (q)

    At node i (performed by the query manager):
        set R_i to the result of performing q on all
            Station Databases on the node that
            are within the scope of the query
        subtract from R_i messages found in HIDE_i
        send R_i to the control node
        delete from HIDE_i all messages for which
            has-seen equals Seq(q)

    At the control node (performed by the node manager):
        send q to each node within the scope of the query
        /*
        * serve other requests until all
        * nodes have returned R_i
        */
        set R_c to the result of performing q on
            the Mailboxes
        subtract from R_c messages found in HIDE
        add to R_c those messages in SEE which
            satisfy q and for which
            has-seen < Seq(q) < will-see
        add to R_c the results from the query managers
            and send R_c to the issuing node
        delete from HIDE all messages for which
            has-seen equals Seq(q)
        delete from SEE all messages for which
            will-see equals Seq(q) + 1
```

VI. AUTOMATIC PROCEDURES

An automatic office procedure [4], [17] captures the notion of an office worker collecting messages at his desk until a "complete set" is compiled. He can then process the messages and file them or send them on their way. By allowing automatic procedures the messages become "active" [16] and require less processing by the user.

The specification of an automatic procedure indicates to the system that it should look for certain messages and act on them appropriately. The specification is nonprocedural. The user indicates what messages are to be collected, and what is to be done with them. He does not specify how they are to be collected or how the actions are to be performed.

Procedures are specified by completing *sketches*. Message templates are used to create, display, and edit sketches; however, a completed sketch is not a message instance. Sketches specify both preconditions and actions. A *precondition sketch* indicates a request to the system to "collect messages which look like this." For example, specifying a pattern in a field of a precondition sketch indicates to the system to collect messages with a field matching the pattern. An *action sketch* indicates a request to operate upon a message that has

already been collected. Simple action operations are to set a field to some constant value or ship a message to a particular station. Action sketches are *triggered* [3], [4], that is, executed, when precondition sketches are matched.

Every message manipulated by an automatic procedure is associated with a precondition sketch and an action sketch. The message's action sketch indicates all insertions and updates to the message. The values to be inserted may be constants, e.g., an authorization, or copied from other fields, or possibly returned from function calls to application programs. We distinguish, therefore, between the original and the updated value of any field. A field which must be copied to another message may itself be modified, and the wrong value must not be used. Furthermore, the function calls may access both the original and updated values of fields. In fact, the original value of a field will often be one of the arguments to the function providing the updated value for the field.

Actions and preconditions may refer to information not found on a message but nevertheless available to the user. These are specified by *pseudosketches* of "pseudomessages." For example, the restriction that a procedure process only messages coming from a particular user is indicated on a special *origin pseudosketch*. Messages may thus be processed differently depending upon their point of origin. Alternatively, the special value not may be used to indicate that only messages coming from stations not listed in the pseudosketch should be processed by the procedure. The special value me is also available to indicate that messages must (or must not) come from within the station's own Station Database.

Actions which do not concern themselves with field values are expressed via *destination pseudosketches*. A common operation is to make a copy of the message being manipulated by the procedure. The message copy to which the destination pseudosketch applies is specified in the pseudosketch's "COPY" field. Copy 0 is the message manipulated by the procedure; copies that have been generated by the procedure are referred to by successive integers. The operations available are leave, ship, and dossier. The first of these requires no argument and simply leaves the message in the Station Database. The second operation transfers the message to the Mailbox database. Finally, the third operation inserts the message into a *dossier*. (A dossier is a named collection of messages, possibly of different types, that are somehow related.) Both ship and dossier take an argument, the name of a station or a dossier, respectively. This may be given as a simple constant or a field or function value, just as in action sketches.

Each active automatic procedure has a *working set* of messages. This is the set of messages manipulated by the procedure and required to activate the procedure. Sketches capture the constraints imposed on the messages in a working set. We may distinguish two classes of such constraints. *Selection constraints* are constant field values, sets or ranges of values, and relations between values of the fields on a single message. Such constraints describe selection criteria on candidate messages for a working set. *Join constraints* are the matching conditions between values of fields appearing on different messages. These describe the join criteria between

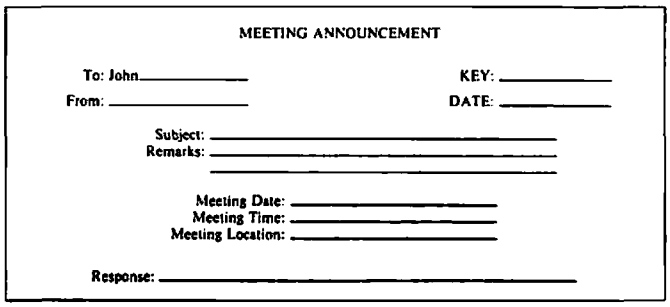

Fig. 5. The *pre_holiday* precondition sketch.

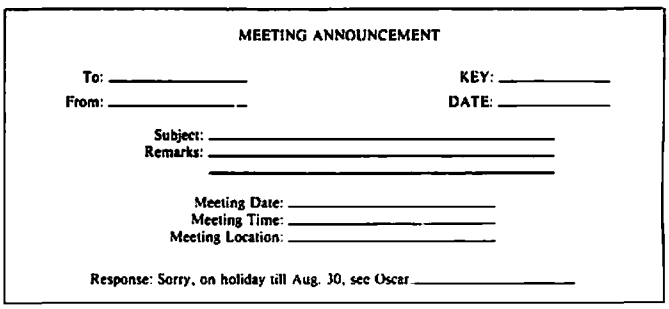

Fig. 6. The *act_holiday* action sketch.

messages within a working set and candidate messages for inclusion in the set. One expects all the messages in a procedure's working set to be linked by certain common field values. Matching field values are therefore probably adequate to model many applications of automatic procedures. However, simple inequality constraints may also be specified.

As a simple example suppose that the user "John" specifies a procedure which replies to meeting announcements while he is away on vacation. The procedure contains a precondition sketch named *pre_holiday*, an action sketch named *act_holiday*, and two destination pseudosketches—*cp0_holiday* and *cp1_holiday*. Fig. 5 shows the *pre_holiday* sketch; this sketch allows only messages sent to "John" to trigger *act_holiday*. This action sketch, shown in Fig. 6, simply causes the value "Sorry, on holiday till Aug. 30, see Oscar" to be entered in the "Response" field. The two destination pseudosketches, *cp0_holiday* and *cp1_holiday*, are shown in Figs. 7 and 8. These are used to return the message and save a copy in a dossier named "missed."

VII. IMPLEMENTATION OF AUTOMATIC PROCEDURES

An automatic procedure in the system is specified by a collection of sketches, and as such describes *what* is to be done rather than how to do it. The sketch representation is very convenient for the user but is unsuitable for processing. The sketches must be parsed and translated for greater runtime efficiency. During translation the legality of actions is checked. The translator also determines whether a legal order of action execution exists. No further runtime analysis of sketches is performed.

Automatic procedures are meant to run regardless of whether the user who specified the procedure has activated his station. The station's automatic procedure interpreter may be activated upon receipt of mail, message creation, or message

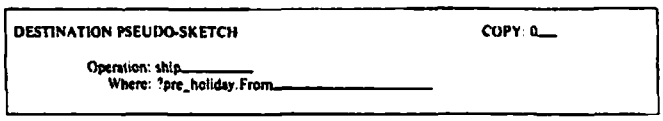

Fig. 7. The *cp0_holiday* destination pseudosketch.

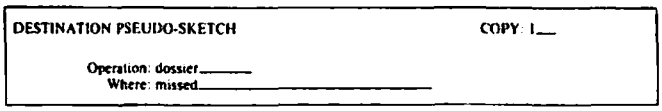

Fig. 8. The *cp1_holiday* destination pseudosketch.

modification. Message creation and modification are performed by the station process and it may then activate the interpreter. In the case of a mail operation, however, it is the node manager which activates the interpreter. Only one interpreter may run at any time for a given station. In this way we eliminate interference problems between interpreters. Once activated, the interpreter communicates with the control node manager, requesting images of messages in the station's Mailbox. These images are maintained in a temporary database of partially completed working sets and are available only to the interpreter.

We cannot predict when the messages required to trigger a message procedure may arrive. The processing must, therefore, of necessity be broken into distinct parts. The specification in terms of sketches contains information of four basic kinds: selection constraints, join constraints, duplicate message type constraints (so that one message is not used to match two sketches within a single working set), and actions. The execution of a automatic procedure makes use of these four kinds of information at different stages.

Suppose that the interpreter is notified of the availability of a message for automatic processing. It first checks whether the message matches the selection constraints of any precondition sketch for that message type. If this is successful and an origin pseudosketch is applicable then the origin restriction condition is tested. If a message does not match the selection constraints of any precondition sketch, then the interpreter assumes that no procedure is prepared to handle it. Suppose that a message does match the selection constraints of one or more precondition sketches. The message is then a candidate for inclusion in the working set for some procedure. At this point a graph searching algorithm [6] is applied to the current working sets. This algorithm determines whether a message may be added by checking join constraints between the message and the working sets.

It is immaterial whether or not a working set is complete since there is always the possibility that at some later time the missing messages may arrive. Even if messages arrive together, the processing of the messages is sequential. The system treats each message individually. A locking algorithm guarantees that two messages cannot be processed at once at a given station. Generally, however, messages will not arrive simultaneously. One can expect a considerable delay between the establishment of selection and join constraints.

Once a working set has been completed the action sketches are executed. Actions are uninterruptible and run to completion. The final step in automatic procedure processing is to execute the destination pseudosketches. This may result in the routing of messages within the working set to other stations and the triggering of procedures at these stations.

The following is an algorithmic summary of the steps followed by an interpreter process:

```
Procedure Interpret( )
    while a message is available for processing do
        obtain an image of the message (either
            from the Mailbox or Station Database)
        if origin and selection constraints are
            satisfied for some procedure
        then
            check join constraints
            if a working set is completed then
                if all messages in the working
                    set are still available
                then
                    obtain the real messages
                    execute action sketches
                    execute pseudodestination
                      sketches
                fi
                cleanup
            fi
        fi
    od
```

VIII. CONCLUDING REMARKS

In this paper we have outlined an experimental system which manages messages. The system manages messages as typed objects which can be sent from station to station. In addition the system enables the user to exploit any message structure to find the messages and to query the information they contain. Finally, the system permits the specification of automatic procedures which are triggered by the presence of messages and manipulate the messages.

The methods of specification of queries, preconditions and actions are similar in our system, i.e., by example. The operations specified, however, are quite different. A query is serviced as soon as it is specified. A precondition results in the system continuously looking for the specified messages so it can trigger a procedure. Finally, an action indicates to the system what it should do with the messages.

Our prototype system differs from most message systems. It uses heavily the informal structure present in messages. It manipulates messages according to content. It handles data inherent in messages in a uniform manner with the message operations. Messages reside in databases. There is a natural way in which information present in them can be tapped for further processing. In this way message processing and data processing can be completely integrated.

Our prototype system differs from most distributed database systems. Most distributed databases assume a complete logical integration of data. Data are distributed physically mainly for performance reasons. In our system there is a logical partitioning of the global database into Station Databases. The concepts of locality and ownership of messages are very

important to the processing of queries. The uniqueness of messages implies that a message is always located at a single station, so there is no replication of data at different stations. The location of a message may also change dynamically, so it is not possible to predict *a priori* where a message resides. This is in contrast to other distributed database systems which often contain much information about the distribution of data items. Finally, updates are restricted to the station at which a message temporarily resides. In this way updates from different stations do not conflict.

Our system is operational [15]; however, changes are being made to the routines for automatic procedure activation and message type definition. We are hoping to expand the system's capabilities in several important directions. First, we hope to integrate more completely the word processing and message capabilities of the system. Second, we are working on multimedia interfaces to the system both for the specification of operations and the presentation of messages. Finally, we are working on the requirements specification, the modeling and analysis of such systems.

REFERENCES

[1] P. Bernstein and N. Goodman, "Fundamental algorithms for concurrency control in distributed database systems," Comput. Corp. Amer., Tech. Rep. CCA-80-05, 1980.

[2] M. S. Broos, S. W. Galley, and A. Vezza, *Data-Based Message Service—User's Manual*. Cambridge, MA: M.I.T. Press, 1978.

[3] J. M. Chang and S. K. Chang, "Database altering techniques," in *SIGMOD Proc.*, Los Angeles, CA, 1980.

[4] P. De Jong, "The system for business automation (SBA): A unified application development system," in *Proc. 1980 IFIP Congr.*, Tokyo, Melbourne, pp. 469–474.

[5] C. Ellis and G. Nutt, "Computer science and office automation," *Comput. Surveys*, vol. 12, no. 1, pp. 27–60, 1980.

[6] J. Hogg, O. Nierstrasz, and D. Tsichritzis, "Form procedures," submitted for publication.

[7] IFIP TC-6 Int. Symp. Comput. Message Syst., Ottawa, Canada, 1981.

[8] P. Kirstein, "New text and message services," in *Proc. 1980 IFIP Congr.*, Tokyo, Melbourne, pp. 521–535.

[9] J. Kornatowski, "The MRS user's manual," Comput. Syst. Res. Group, Univ. Toronto, 1979.

[10] I. Ladd and D. Tsichritzis, "An office form flow model," in *Proc. NCC*, Anaheim, CA, 1980.

[11] D. Luo and S. Yao, "Form operation by example—A language for office information processing," in *SIGMOD Proc.*, Ann Arbor, MI, 1981.

[12] C. Papadimitriou, "Serializability of concurrent database updates," *J. Ass. Comput. Mach.*, vol. 26, no. 4, pp. 631–653, 1979.

[13] F. Rabitti, "Distributed query facilities for office information systems," Univ. Toronto, Tech. Rep. CSRG-127, pp. 201–234, 1981.

[14] D. Tsichritzis, "OFS: An integrated form management system," in *Proc. 1980 VLDB Conf.*, Montreal, pp. 161–165.

[15] ——, "Omega Alpha," Univ. Toronto, Tech. Rep. CSRG-127, 1981.

[16] J. Vittal, "Active message processing: Messages as messengers," presented at the IFIP TC-6 Int. Symp. Comput. Message Syst., Ottawa, Canada, 1981.

[17] M. Zisman, "Representation, specification and automation of office procedures," Ph.D. dissertation, Wharton School, Univ. Pennsylvania, 1977.

[18] M. Zloof, "Query by example: A data base language," *IBM Syst. J.*, vol. 16, no. 4, pp. 324–343, 1977.

[19] ——, "A language for office and business automation," IBM T. J. Watson Res. Center, Tech. Rep. RC8091, 1980.

Dennis Tsichritzis received the Diploma in engineering from the National Technical University of Athens, Athens, Greece, in 1965, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, both in computer science, in 1967 and 1968, respectively.

He has widely published in different areas of computer science including operating systems and database management systems. His current interests are in the area of database management systems, data models, and office information systems. He is presently with the Computer Systems Research Group, University of Toronto, Toronto, Ont., Canada.

★

Fausto A. Rabitti received the Doctor's degree in computer science from the University of Pisa, Pisa, Italy, in 1976.

After receiving his doctorate he obtained a fellowship at the Information Processing Institute (IEI) of Pisa for research on centralized/distributed operating systems and computer communication systems. Since 1980 he has been at the Computer Systems Research Group of the University of Toronto, Toronto, Ont., Canada, with a two-year fellowship for research in the field of distributed databases with applications to office information systems.

★

Simon Gibbs received the B.Sc. degree in physics from the University of Alberta, Edmonton, Alta., Canada, in 1976, the M.Sc. degree in physics from McGill University, Montreal, P. Q., Canada, in 1978, and the M.Sc. degree in computer science from the University of Toronto, Toronto, Ont., Canada, in 1979.

Since 1979, he has been with the Computer Systems Research Group, University of Toronto, and is currently working towards the Ph.D. degree. His research interests include data models and their use within office information systems.

★

Oscar Nierstrasz was born in The Netherlands in 1957. He has spent most of his life in Canada. He developed an early enthusiasm for computers at his high school in Toronto, but he spent his undergraduate years studying pure mathematics and combinatorics at the University of Waterloo, Waterloo, Ont., Canada. After coming to the University of Toronto, Toronto, Ont., he became interested in computers in automation, writing his Master's thesis on a system for writing automatically triggered office procedures. He is currently doing his doctoral research at Toronto in modeling and analysis of information flow in an electronic office network.

★

John Hogg received the B.Sc. and M.Sc. degrees in computer science from the University of Toronto, Toronto, Ont., Canada, in 1978 and 1981, respectively.

He is presently working toward the Ph.D. degree at the University of Toronto. His interests include office automation and local networks.