Identification

MIT-DMS Communication System Overview

Jack Haverty

May 15, 1975

Motivation

An overview of the current MIT-DMS communication system is contained in this document. The system is intended to handle all of the common uses of a message handling system, as well as to provide a powerful base for more sophisticated uses. This memo describes the underlying design philosophy of the system. It presents the concept of a message as a collection of named attributes, and the data accessing and processing model used implement the current system. Included is a description of the model of access primitives for a message data base, and a description of the capabilities which can be implemented in a message system based on the model. Details of the actual implementation are found in the references.

References

Black, Edward H., The DMS Message Composer, Project MAC, Programming Technology Division Document SYS.16.02.

Broos, Michael, IRS -- MUDDLE's Information Retrieval System, Project MAC, Programming Technology Division Document SYS.11.17.

Haverty, Jack, Communications System Daemon Manual, Project MAC, Programming Technology Division Document SYS.16.01.

Pfister, Greg, A MUDDLE Primer?, Project MAC, Programming Technology Division Document SYS.11.01

Table of Contents

I. Introduction

I.1. Scope of System

This document is intended to summarize the organization and philosophy involved in a communication facility suited for network use. This facility was developed both to provide a powerful message handling system to a community of users, and, more importantly, to formulate a model of the structure of a message handling system to support construction of sophisticated message processing environments.

The logical structure of the system, which is more useful in evaluating complex systems, is described here instead of the details of the implementation. Details of the current implementation and use are to be found in the references. The system is designed to function in an environment such as the ARPANET provides, a network of server computers linked by medium-bandwidth communication lines. The users of such a network are assumed to have one or a few primary sites where they may maintain message bins for receipt of messages. Users of the system are not restricted to be people; processes can easily use the system to communicate, especially since the messages are maintained in a structured, machine-readable format at all times. The types of messages handled can range from short notes to large formal documents. Such considerations affect the design of the various processes involved, and the architecture of the processing facilities

themselves. The messages are, however, intended to be not to be necessarily handled in a 'real-time' manner, as would be needed for teleconferencing and similar applications. Transmission times can range from seconds to minutes, depending on the load of systems involved, etc.

The system as it is commonly used may be viewed as composed of two logically distinct parts. One of these is the user interface, which is inherently interactive. The other is a background processing facility, to which users may assign some or all of the processing associated with a message, to be done by an 'absentee' process. Generally, all non-interactive processing is delegated to the background facility, permitting the user to resume other tasks. The foundation for both parts is a data base and associated accessing primitives, which are used by all functions to perform a task for a message, as well as by the daemon's background scheduler, which handles applying processes to messages in an absentee job.

This memo concentrates mainly on a description of the data base and daemon processing facilities, with only a short description of the characteristics of the interactive parts of the system. Further details on the user interface can be found in the references.

I.2. Data Base Model of a Message

In communications between humans as well as between processes, it is often useful to think of a message as a structured object, composed of a collection of attributes of some abstract entity which is the message. For example, some attributes of a message are its text, addressees, author, time written, time delivered, subject, etc. The facility developed during this project is based on the concept of handling of a message as a collection of attributes, whose identity is maintained during the entire handling process, instead of the simpler model of a body of text which is delivered to a list of people. The attributes of a message are identified by a mnaemonic name. In all documentation concerning the system, the value of some attribute is referred to as a 'field' of a message, e.g. the text field.

The treatment of a message as such a structure facilitates the construction of message-handling systems which are in some sense intelligent, since the data defining the message is easily kept in a machine-processable format.

This organization enables independent functions to be created to perform various operations on a message without interference, since they can use a private set of fields to contain needed data. Functions can be written to communicate using mutually-known fields. In fact, the standard sequence of operations to handle transmission and delivery of messages in the 'normal' fashion

is implemented as a set of such cooperating functions. Handling and storage of messages in this structured form, and adherence to the philosophy of organization of message-handling facilities as a set of co-operating processes enables the message facility to provide a number of powerful operations which the author and receivers of a message can apply or which they can cause the daemon to apply for them.

In summary, the system can be viewed as consisting mainly of a data base manager, which handles objects consisting of sets of named fields, a set of functions, which operate on those objects to accomplish some operation, and a scheduler for such functions, which handles applying them to indicated messages in a background job for an absentee user.

I.3. Functions and the Scheduler

Virtually all operations which are done to a message are performed by a function. Functions are written to conform to a few simple rules for operation, such as always accessing message fields using the supplied data-accessing package. Functions may be applied to a message as part of some interactive request, or they may be automatically applied by the daemon scheduler at the request of some user, which is the more common occurrence.

The scheduler includes the capability to apply functions to particular messages after a specified point in the future. The protocol used by the scheduler to apply

functions to a message also includes the ability for a function to indicate that it has 'failed' in a temporary fashion, i.e., if tried later it may be successful. The scheduler in such instances automatically retries the function at intervals until it succeeds or a predefined number of attempts fail.

The scheduler protocol includes an error-handling facility, which provides for sending a message to the owner of a message when a function returns a fatal failure indication. Such a message is composed by the scheduler, using a default error analyzer, or optionally one written specifically to analyze errors in the particular function. In addition, the analyzers for current functions are capable of detecting and reporting probable bugs to the maintainer of the communication system.

The particular functions which have been written in this environment handle message communication, but such a restriction is not inherent in the structure.

I.4. Control of Functions by Message Users

Because the individual attributes of a message are separable, and remain computer-processable throughout the message's lifetime, essentially any algorithmic, or even heuristic, operation can be implemented to process a message, at any point in its travels. The model which has been developed, and the resulting implementation, provide at least two places in the normal message handling where such control can occur, one which handles operations for the

author, and another for the receiver. These places occur where a particular function is applied which interprets a set of instructions obtained from a message field. The instructions constitute a program which is executed by the interpreter function to accomplish whatever the associated user has programmed. This facility essentially provides a means for individual users of the message system to tailor its operation according to their own personal desires.

Generally the user of a message will have his personal program simply select and schedule some standard functions to be applied to his message. The facility is extensible however, in that any user may write a function, conforming to the defined protocol, place it in the program library, and direct the scheduler to apply it to the message. In this manner, highly personalized functions may be added to the system in a non-disruptive fashion, since no existing software needs to be changed.

Often a recipient of a particular message desires some action to occur as a result of that message's arrival. In conventional message systems, the user must actually read a message and manually initiate whatever operation is desired. This can involve operations as simple as printing the message on a line-printer to any number of more complex decisions such as forwarding the message to other interested parties. In the model of the message facility, the other control point, where the receiver of the message can specify processing to occur, is triggered when a message arrives for the specified receiver. This is accomplished by

including in the set of normal functions which are applied during handling of an incoming message a function which interprets instructions obtained from a standard field. This enables a receiver to cause action to occur as a result of an incoming message immediately on its arrival, which may be long before he actually reads it. Since the message is structured, the instructions can examine the message components and make decisions based on the contents. This facility provides an environment where a user with a message bin can effectively have a personalized process running at all times, waiting for messages; such a daemon resides in the instructions which are provided for action to be done on arrival of a message.

This structure also enables a similar function, to which the author supplies instructions via a message field, to be applied whenever he creates a message. For example, this would permit construction of a powerful office environment as a collection of functions, where the various functions could perform such tasks as spelling correction, formatting of the message, and coordination before the actual transmission, holding up the message and informing the author only if an error occurs. In addition, it enables the author to set up a personalized environment for his messages, which can be 'intelligent' in some sense. The function can be used to tailor the contents and handling of individual messages according to any criteria which are programmable. For example, a fairly complex function could examine the text of the message, and add entries to the distribution list depending on the contents. Arbitrarily complicated processes can be imagined, such as ones which

perform different functions depending on the day of the week, urgency of the message, etc.

I.5. Timing of Functions

The daemon's scheduler implements a facility for specifying that a function be applied to the message at some specific point in the future. The scheduler attempts to apply the message at the stated time. This capability is used to implement various actions which require such memory. Several example uses of this capability follow.

The author of a message often would like to be informed of the success or failure of delivery of his message. He may, for example, want to be informed if one or more addressees have not received the message by a certain time, or have not actually read it by some time. This facility is implemented by a function which schedules itself to run at the next critical time, to check if the author should be informed that some event has, or has not, occurred. The function is general in scope. It operates with attributes of the message called 'conditions' and can maintain a table for each addressee of action to take when a condition occurs, or if it has not occurred by some specified time. Several conditions are currently used, among them 'delivery' and 'reading'. Implementation of such capabilities depends on the ability to schedule processes to be run at specific times in the future or on occurrence of event associated with a condition. The

former is handled by the time specifications of the scheduler, and the latter is obtained by an interrupt-like facility which is part of the function application protocol.

Especially in operation in a network environment, a message system should handle as much of the tedium of getting the message through to the receiver as possible. For example, this includes handling messages intended for a site which is currently unavailable. The author of a message should not be required to wait while a message is transmitted, or be forced to retry manually if the receiver's site is unavailable, but rather should be capable of specifying how a message should be handled, and have the daemon keep trying to get the message through to the receiver. This capability is also handled by the delivery function's ability to reschedule itself to run later, and to keep data in private message fields to use as a history of its attempts, so that it has the ability to 'give up' if matters appear hopeless, generating an error which the standard mechanisms will report to the owner of the message. This capability is also a result of the architecture of the data base accessing primitives and background process scheduler, described in detail later.

II. Data Base Model of a Message

II.1. Message as a Set of Fields

The message data base contains a set of objects called messages, which have attributes identified by names, e.g. text, to, blind, carbon-copy, from, etc. The value of any attribute is referred to as the contents of the relevant field of the message in some user's area. The data base's operational characteristics are defined by the set of accessing primitives supplied to users of the data base. The most common users are the communication daemon itself, processes which the daemon applies to a message on some user's behalf, and various interactive processes such as the Reader, Composer, etc.

All users of a message have an assigned area of the message in which they can read and write data in fields. Each area is accessed by specifying the name of the related user. Each receiver of a message is a user, who accesses his area by specifying his name, and the author of the message is a user, who accesses his area by the special 'empty' name. A user can be both the author and receiver of a message, with different associated areas, using this convention. In general, different user areas are completely independent; the same field may contain different values, depending on whose area it is obtained from. This enables use of fields such as 'when-delivered', which can be different for the various receivers. The number of fields which a message can contain is not limited.

Functions can be designed to use fields in their operation which did not previously exist, and which will exist only for messages which utilize the particular function. There is, however, no protection from naming conflicts between functions, but this has not proven to be a problem, since most functions must be approved for public use within the daemon, at which point conflicts can be resolved. Personal functions can easily be written to use names based on the person using them in a unique manner.

II.2. Sharing of Data among Users

In general, any specific user of a message can only access the fields in his personal area. A provision for controlled data sharing is included, to permit receivers to access selected fields from the author's area, to obtain the delivery time, for example, and vice-versa, to obtain the text, for example. Such access is controlled for purposes of privacy. In each user's area is a field which controls access to his area from external users.

The effect of the sharing mechanism is to create for each user of a message a pseudo-area which is composed of the user's own area plus selected parts of one or more other areas. The author sees the message as containing his own area, plus one area for each receiver, containing fields to which the receiver has permitted access. Each receiver of the message sees a message data base which contains two areas, his own and parts of the author's area. The area, however,

contains only those fields from the author's area which have been specified in the author's access control field as being available to the particular receiver.

The author has complete control over everything in his area. He is, for example, the only one who can change any item in his area, such as the text. In addition, he can selectively enable addressees to read specified fields of his area, such as the text. Such fields appear to the addressee as if they were really contained in the addressee's area. The only way an addressee can read the value of an attribute of a message stored in the author area is if the author gives him access to it. An attribute called 'notes', for example, where an author stores personal reminders related to the message, could be made inaccessible to any particular addressee.

Conversely, the individual receiver's areas could contain items such as the time delivered, personal notes about their comments on the message, etc. Access to these fields is totally controlled by the receivers, who can specify which fields the author is permitted to examine. The author cannot change any receiver's fields with the data accessing primitives. The author sees the message data base as containing several areas. However, the various receiver's areas will appear to contain only fields to which the author is permitted access. The capability of an author modifying a receiver's area is embedded only in an internal daemon process which actually handles the creation of the receiver areas when the message is sent.

From each receiver's viewpoint, the message initially consists of a set of attributes such as the time delivered, which the message transmission function automatically generates, and also all attributes which the author has permitted him to access. These fields are virtually in the receiver's area, i.e. he accesses them by requesting the given field from his own area, instead of the author's. These fields will appear to have the value which the author specified for the particular addressee, but they are inherently read-only fields. If the receiver writes data into the field, it will be written into his own area, overriding the value from the author's area. This capability is useful to override such fields as ones containing specifications of output formats to be used, when the individual receiver does not want the author's recommended specification to be used. For example, transmission of graphics information may include a field from the author area specifying a format of output which is unavailable or undesirable for a particular receiver, who can override the author's specifications by writing his own value into the field.

The receiver's area is automatically created when a message arrives. Note that the attributes passed through by the author are not necessarily copied into the receiver's data base, but are obtained from the author's area by the data accessing primitives. This can be of significant practical importance when large messages are handled involving many addressees. The data base accessing is handled in a way that permits receivers to read such attributes as if they were in

the receiver's data area, when they are actually being retrieved for all receivers from the author area.

The receiver can create attributes at any time for an existing message. Generally this would be done by some function which the receiver has caused to be applied to the message, or by issuing a command while using the reader subsystem. This enables him, for example, to examine the message, compose a reply, and store the identity of the reply message in the received message area, for use in later information retrieval. He could also, for example, compose notes and ideas as he reads the message, and store them as an attribute of the message for later retrieval.

II.3. Data Field Defaulting

Many of the fields of a message are used to supply instructions to the various functions to control their operation. For example, the delivery function accesses a field to obtain a specification of where to output a message, what fields to output, and the format of output to use for each field. The addressee expansion function, which maps names of groups of addressees into the equivalent lists of individuals, uses a field to obtain a list of data bases to use in its processing, and so on. Inclusion of such fields in every message would be prohibitively expensive, since some of the fields contain large amounts of data, and often they are identical for many messages.

The data accessing primitives implement a search scheme for fields, which permits maintenance of sets of values to be used for each field as defaults, if no such field exists in the individual message being processed. Such a collection of defaults is termed a 'tailor data base', since it is used to tailor the various processes of the system to individual user's preferences. The data accessing primitives use the library system to obtain the data base for the user currently being processed. Whenever a field is requested which is not in the message area itself for that user, the user's tailor data base is scanned to see if a default value is provided.

In the case of a receiver's area, the data accessing process continues, if the field does not exist in the user's area or tailor data base, and examines the author's area of the message. If the specified field is not found there, the author's tailor data base is scanned.

The last resort, in all cases, is to examine the 'system' tailor data base. The common specifications for the various handling processes is contained in this data base. The normal characteristics of the various processes are controlled by the values of the relevant fields in the system tailor data base, instead of being hard-wired into the code. In this manner, the default characteristics are easily modified.

II.4. Multi-valued Data fields

The data base accessing primitives allow for multiple-valued attributes in the author's area. One or more fields of the message may be set up by the author to appear to have different contents depending on which receiver is reading it. This is useful for such data as carbon-copy lists, comments, etc. In the case of a comments field, for example, the author could store notes personalized to the individual addressees. The individual receivers see only the value specified for them.

This facility is totally contained in the data accessing primitives, which handle selection of the appropriate value for such a field when a request to read is being performed for some receiver of the message in the author's area.

III. Structure of Daemon Processor

III.1. General Organization

The architecture of the communication facility was determined by several basic design goals. First, the system was meant to be easily usable by both human users and processes. Second, the user should not be required to wait needlessly while the message transmission was accomplished; only processing to support

interactive facilities, such as editting the text of a message, needs to be done while the user waits. Third, the system should be capable of being "programmed" to a large extent. Users can set up their particular entries in the data base to control how the message system behaves for them, both as authors and recipients of messages. This is especially valuable for experimentation purposes.

The communication facility can be logically divided into two distinct cooperating parts, the background processing facility with its various functions, and the user interface.

The first part is composed of the daemon background scheduler, which applies various functions to a message at specified times. The customary operation of sending of a message is accomplished by the sequential application of these functions to the message; this is generally handled by the daemon, but can be done while the user waits, if he desires.

The user interface is comprised of a composer, reader, and information retrieval system, in a single uniform environment. This part is generally used only by human users; functions presumably do not need the facilities provided, or, if they do, the function can implement the required operation itself, or, more commonly, call the same subroutines which the interactive processes call from the system library.

The interactive systems communicate with the daemon facility via the message data base accessing primitives, as well as several mutually-recognized files.

III.2. Daemon Organization

The communication daemon (referred to as "the daemon" from now on) generally handles virtually all of the non-interactive processing involved in processing a message. It accesses the message data base through the standard data accessing primitives. In addition, it maintains a data base of messages needing processing, to facilitate scheduling of functions at requested times. It also watches for the existence of various 'request' files, generally created by other jobs, which contain specifications of new messages to be created, new processing to schedule, etc.

The daemon operates as a system background job. It has the ability to specify to the operating system when it should be run next. Normally, the daemon schedules itself to be run at the next time it has a message needing some action, or every few hours, to scan for possible input files which have been created in the interim. In addition, processes which create such input files may signal the operating system to run the daemon immediately, to handle high-priority requests.

The foundation of the daemon is the scheduler, which is responsible for running of the requested functions for various messages as soon as possible after the time specified. To accomplish this it maintains a data base of messages needing action, and the earliest time specified for any of the future actions. The message itself is used as repository for the detailed information about the processes to be run, in a standard set of fields reserved for that purpose.

III.3. Data Base Access Protocol

The daemon, like all jobs in the message handling system, accesses messages via the standard data base primitives. The basic unit record of information used in the communication system is the message. The data accessing primitives are designed to operate on only one message at a time. Associated with each message, throughout the system processes, is a lock, which the data accessing primitives attempt to lock before accessing any message. The lock can be soft-locked, in which case the data accessing functions are inherently read-only. More commonly, a message's lock may be hard-locked, in which case the data accessing primitives can both read and write fields of the message. The operating system assures that only one job may have a message hard-locked at any time, while any number of jobs may soft-lock a message.

Generally, a job wishing to process a message, such as the daemon's scheduler, tries to hard-lock the associated lock, and if successful, does its work and unlocks the lock. The data accessing primitives are set up so that changes to a message do not actually get permanently recorded until the unlocking operation is accomplished. This manner of operation insures that a message will not be left in an inconsistent state, since application of functions to a message is effectively an atomic operation, i.e., either it finishes completely, or nothing is done.

The scheduler attempts to lock a message when the time arrives at which it

remembers a function is scheduled to be run. If the locking is unsuccessful, possibly because another job is accessing the message, the daemon reschedules another attempt for some time in the future. If the locking succeeds, the daemon examines the various message fields which contain specifications of which functions to apply, applies the functions, and then checks to see when the next function is scheduled. The scheduler's data base is then updated to indicate when this message should be handled again, the message is unlocked, and the daemon goes on to its next task.

This method of operation virtually assures against inconsistency in the data bases, or loss of messages due to crashes. Since the message or the daemon's scheduler data base is not updated until the processing is complete, a crash at any point will at worst cause some function(s) to be repeated partially. This should be taken into account when functions are written to run under the daemon's scheduler.

III.4. Function Application Conventions

Functions which are written to be run by the daemon scheduler must conform to a simple protocol.

To access a message, they must use the standard data accessing primitives. When they are applied, a message will already have been hard-locked by the daemon. The function must work only with that message, and return with it still

locked, so that the daemon may update the various function control fields of the message. The same conventions are followed by programs, such as the composer, which run as user jobs.

The function must be written in such a fashion that it is resumable. If the daemon or system crashes during the processing, and the function is reapplied later, it should be able to recover gracefully. Generally this means that the function must not do things like deleting files, etc. which may be needed for a successful later application of the function.

If the function decides that it should run later, it can either return an error code indicating that a non-fatal error has occurred, or, optionally, explicitly schedule itself for some future time, and perform a successful return. In the latter case, the function must be carefully written to avoid infinite repetition if the error is never corrected. In the case where an error code is returned, the function will be prevented from running by the scheduler, after a number of unsuccessful tries.

In most cases, an error analyzer should be written to be used by the daemon in deciding whether error codes are fatal or non-fatal, and in composing a message to send to the owner of a message when a function fails completely.

The function, since it runs within a daemon performing tasks for many users, should be reasonably friendly about limiting its run-time, avoiding creation of fatal conditions within the job, etc. Most functions will probably have to be 'approved'

by the system maintainer before they are permitted to be run by the daemon scheduler.

These restrictions have proved in practice to be relatively painless in programming the various functions available now within the communicaion system. In fact, by limiting the possible external interactions between the function and the outside world, a more reliable and modular system has been obtained.

III.5. Message Conditions and Signalling

The previous sections describe how a function may explicitly schedule another function, or itself, to run at a specific time. In the case of several capabilities which are desired in a sophisticated message handling system, it is necessary to provide a mechanism for running a function when some condition occurs. For example, a function which is intended to report that a message has been delivered to some addressee should be run when the delivery process succeeds for that addressee. In general, there may be several processes which should be run when something happens.

The mechanism which is used to handle these cases strongly resembles a standard interrupt facility such as exists in many programming languages. Processes which cause some state of events to occur can inform whoever is interested by using a system primitive called signalling. Conditions are identified by mnaemonic names. For example, when the delivery process is successful, it

signals that the condition 'delivery' exists for the message. Any function which desired to run again on occurrence of a condition specifies the condition to the daemon scheduler system, using a supplied primitive. Whenever any condition occurs, the functions which have requested to be signalled are added to the related message's queue of functions to be applied.

This facility completes the symmetry of scheduling capabilities in the daemon. The author of a message can cause processing to begin for an addressee by running the transmission function, which is the only function runnable by an author and capable of scheduling actions for a receiver. Any user can, of course, schedule functions for his own area. The signalling facility enables a receiver of a message to cause functions to be scheduled for the author.

III.6. File Input to the Daemon

The majority of programs interfacing to the communication system use the standard data accessing primitives to modify message fields. An alternative input path is provided for additional flexibility, and to enable programs not resident in the MUDDLE environment to use the communication system facilities.

The daemon supports a file-input path, which treats files of publicized names as command files for the daemon. The most common use of such files is to create a new message, and schedule some functions to be applied to it, usually to accomplish the normal sending of the message. The file input syntax supports

creation of new messages, writing of data into fields, and scheduling of functions to be applied to the message. The functions available are operationally identical to those provided in the data accessing and scheduling primitives available in the normal programming environment, and in fact are implemented by an interface between the file parser and those primitives. Obviously only write-type operations can be supported in this fashion, but these are the most common functions needed to cause the system to perform some action.

Commands are not restricted to new messages. The input file can specify an existing message and user to be accessed, and the daemon will automatically keep trying to lock the specified message and, when successful, carry out the commands.

In addition, the file input path provides an escape hatch for interactive processes which desire to modify some message which is currently locked by another job. In this case, the process can simply write out an appropriate command file, to be handled by the daemon later.

The file input syntax and repertoire represent a rudimentary beginning of the definition of a standard protocol for communication between multiple instances of daemons, probably at different network sites, to handle messages in a structured fashion, and support process scheduling, on a multi-site basis.

IV. Interactive Systems

IV.1. Introduction

This chapter is intended to give the flavor of facilities provided by the user interface parts of the communication system. The references contain more complete descriptions of the exact structure of the interface, and repertoire of capabilities installed within it.

The interactive parts of the system are intended for use by humans to handle messages to other humans or processes. It serves to provide an environment for reading, writing and editting messages, specifying how a message is to be handled, and sending messages. Additionally, commands are provided to read incoming messages, reply to them, and examine the data base of all past messages which the user has sent or received. The commands are available simultaneously, which provides an environment in which a user can read a message, look up some old message in his data base, send a reply to the message, forward it, compose a new message, etc., in any order.

The system also provides commands to enable the user to set up his own personalizing specifications in the daemon's data bases, to cause it to handle his messages as he wishes. Additionally, it contains commands to query the status of messages previously sent, to determine if they were delivered, for example, or to retrieve information about them for use in composing a new message (e.g. a reply to a reply concerning the message).

IV.2. Composer

The composer commands implement facilities useful when creating a message. Individual fields of a message, such as the text, addressee list, subject, etc. may be input, examined, or editted at any time, in any order. Various editting facilities are provided, including a capability for editting field contents, such as the text, using one of several standard system text editors. The message is finalized and sent when the 'send' command is issued. If the user specifies that the message be saved in his data base, he can later retrieve the message, edit its fields, and resend or refile it, for example. This permits a facility for a recipient to add his own keywords or to cause the message to be filed under a portfolio name of his own choosing, for example, to be accessed in the future using the information system commands.

The composer can also, through an interface to the user's data base, retrieve messages which he has received, so that he may edit and resend them, to provide a forwarding facility.

IV.3. Reader

The reader commands are intended for interactive use by humans to examine and act on the messages they have received. It provides indexing facilities for categorizing messages according to their urgency, author, etc., and commands to

specify how to dispose of the messages. The daemon provides several basic processing capabilities which the reader can trigger, such as printer output, file output, etc, which are provided as reader commands. Additionally the reader system keeps track of the status of each message, so that the user can inquire at any time to find out what messages are still pending, need replies, etc.

The reader and composer commands are logically distinct, but they are generally used concurrently, possibly in conjunction with the information retrieval system facilities also. The interface between the three is sufficiently powerful that a user can treat the entire system as a single communication handling system, instead of separate programs.

IV.4. Information Retrieval System

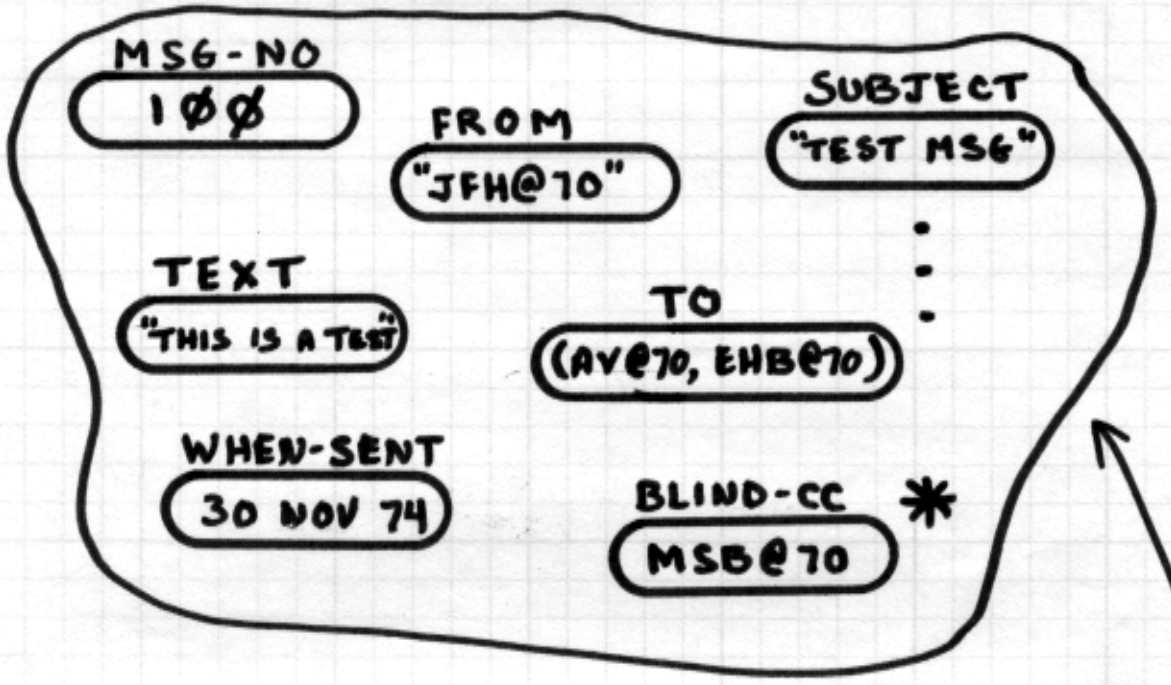
The information retrieval commands form a generalized data base interrogation system, which is not strictly a part of the communication facility, but which was developed separately for use in subroutine library systems. It has been interfaced to the communication system however, so that messages can now be entered into one or more data bases, and the retrieval system commands used to examine the messages contained. For example, a user may have all of his received mail placed in such a data base automatically, so that he can, at any time, examine old messages, sort them according to author, etc. A smart personal function specification could easily control which of many data bases a given

message was inserted into based on the subject, keywords, or anything else it can examine when applied to a new incoming message.

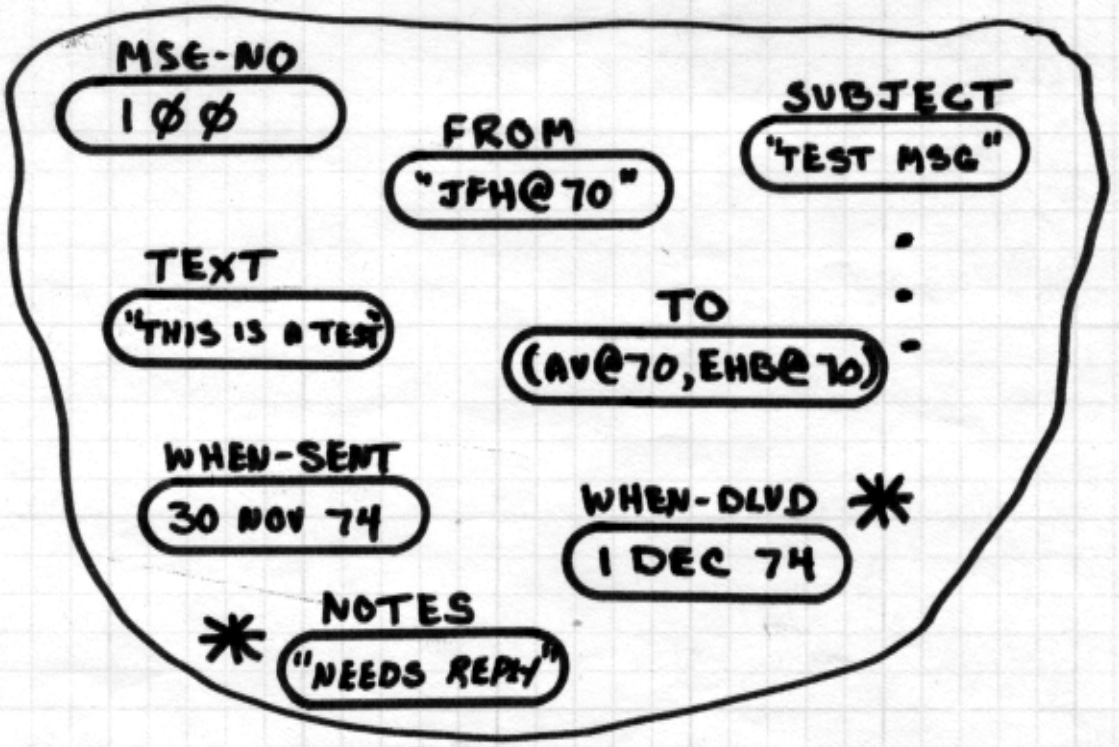
A user would typically use the information system commands to search his data base for a specific message, and then use the composer commands to retrieve that message for editting and resending.

The three interactive components are usable simultaneously, so that the user can easily use functions of any system without inconvenience. This would permit, for example, a user to read his mail, retrieve a message to which he has just received a reply, and refer to it as he composes a new message to reply to the one just received.

MODEL OF MESSAGE: A SET OF FIELDS WITH NAMES & VALUES



a) A MESSAGE, AS SEEN BY ITS AUTHOR



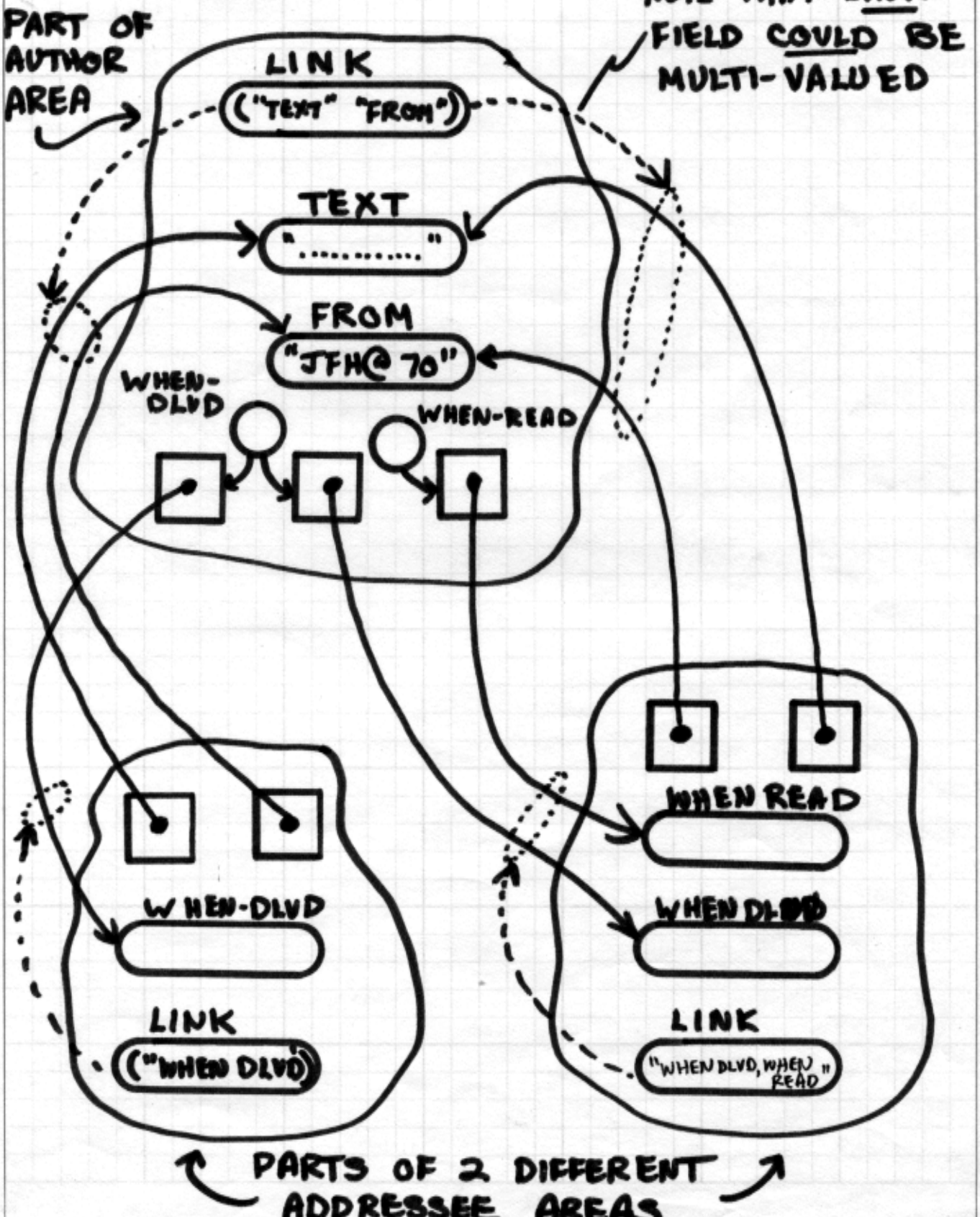
b) SAME MESSAGE, AS SEEN BY ONE ADDRESSEE

TYPICAL FIELDS ARE SHOWN

* ⇒ FIELD WHICH HAS DIFFERENT VALUES

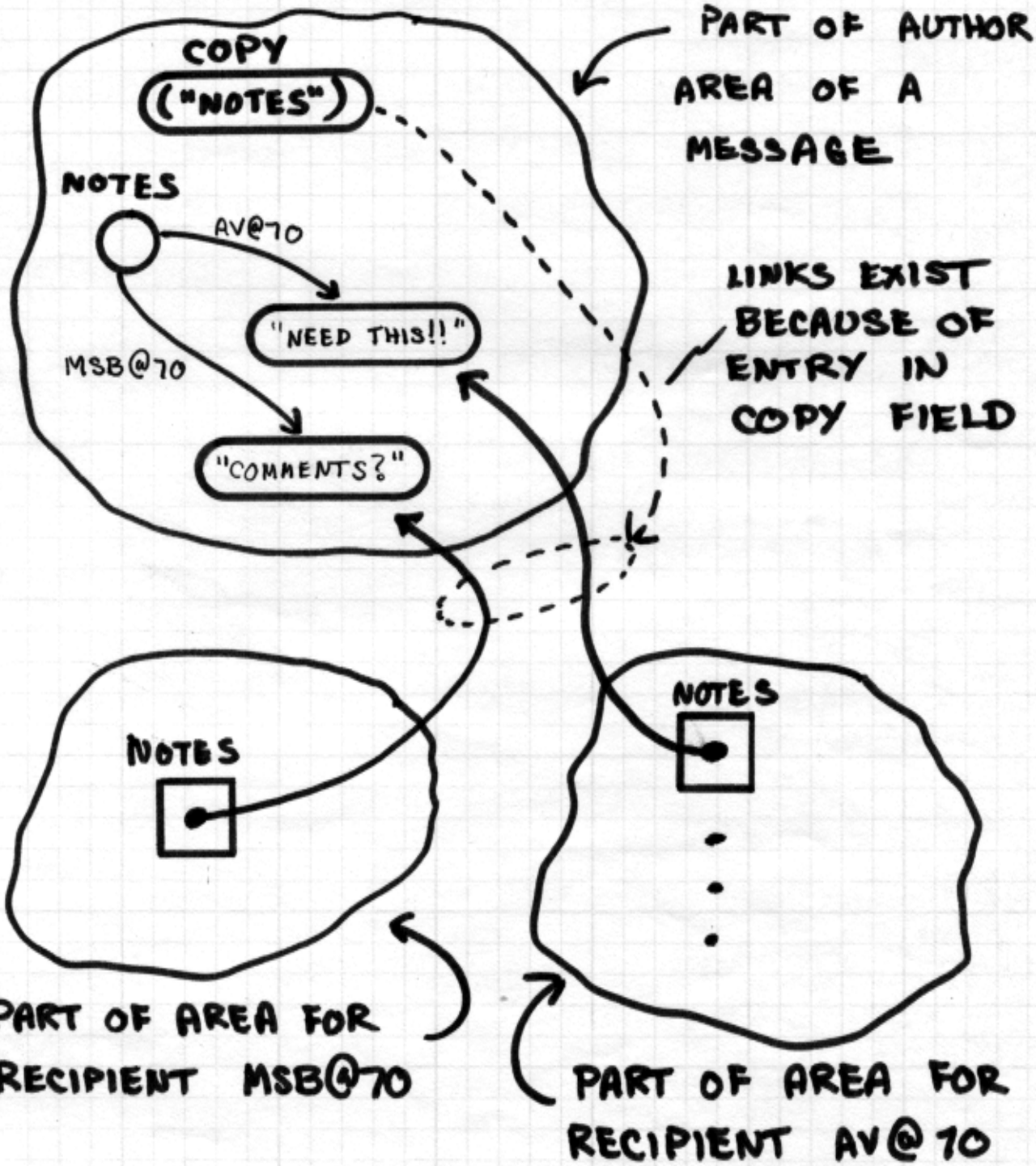
MODEL OF MESSAGE: DATA SHARING/LINKING

PURPOSE: TO REDUCE DUPLICATION OF STORAGE AND PROVIDE FEEDBACK



MODEL OF MESSAGE: MULTI-VALUED FIELDS

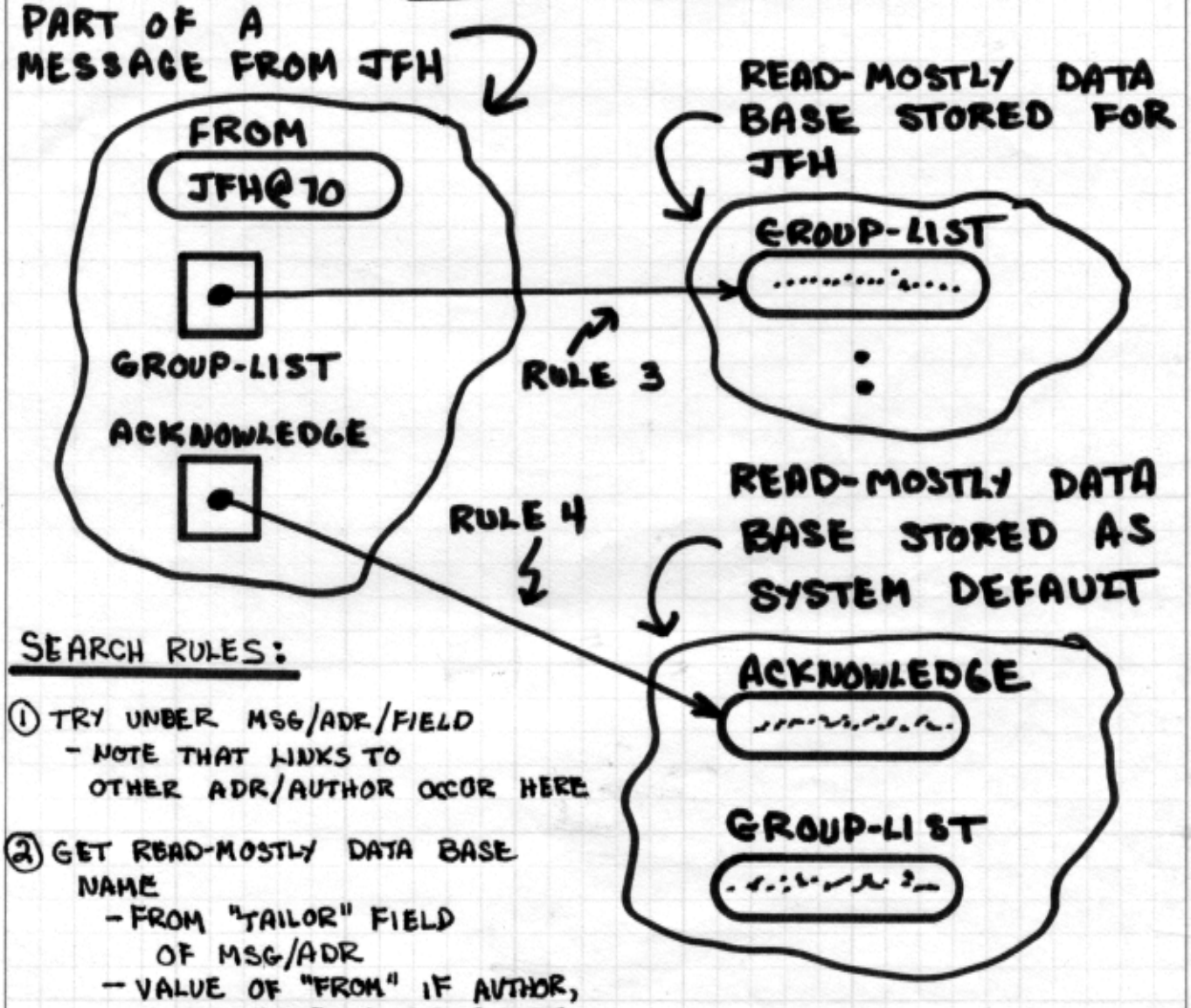
EFFECT: VALUE OF A FIELD IN THE AUTHOR AREA IS DEPENDENT ON WHO IS READING IT.



- NOTE THAT RECIPIENT EHB@70 SEES NO NOTES

MODEL OF MESSAGE : DEFAULTING FIELD VALUES

PURPOSE - REDUCES STORAGE & PROCESSING FOR FIELDS WHICH USUALLY HAVE A CONSTANT VALUE FOR MANY MESSAGES



SEARCH RULES:

① TRY UNDER MSG/ADR/FIELD
- NOTE THAT LINKS TO OTHER ADR/AUTHOR OCCUR HERE

② GET READ-MOSTLY DATA BASE NAME
- FROM "TAILOR" FIELD OF MSG/ADR
- VALUE OF "FROM" IF AUTHOR, ELSE NAME OF RECIPIENT

③ TRY UNDER FIELD NAME IN DATA BASE IDENTIFIED IN ②

④ TRY UNDER SYSTEM DEFAULT DATA BASE

PROCESSING OF MESSAGES

A) OPERATIONS ON MESSAGES ARE MODULARIZED INTO NUMEROUS PROCESSES, WHICH USE SYSTEM PRIMITIVES TO ACCESS THE MESSAGE DATA BASE, FOLLOWING A SET OF CONVENTIONS IN THEIR ALGORITHMS.
   - EASY TO ADD NEW PROCESSES
   - EASY TO ALTER THE SEQUENCE & TIMING OF THE PROCESSES A MESSAGE UNDERGOES

B) PROCESSES MAY BE APPLIED TO A MESSAGE BY ANY USER, INTERACTIVELY, OR MAY BE RELEGATED TO A DEMON FOR BACKGROUND PROCESSING.
   - MODEL EXTENDS TRIVIALLY TO MULTI-PROCESSOR ENVIRONMENT

PROCESS ENVIRONMENT & RULES

(1) PROCESSES ARE GIVEN A MESSAGE # AND USER NAME, TO USE IN ACCESSING A PARTICULAR AREA OF THE DATA BASE. IN MOST CASES, THEY SHOULD OBTAIN ALL DATA TO ACCOMPLISH THEIR TASK VIA THE DATA BASE PRIMITIVES

(2) PROCESSES MAY SCHEDULE OTHER PROCESSES BY MODIFYING THE DATA AREA'S 'PROCESSING-NEEDED' FIELD

(3) PROCESSES MAY BE SUCCESSFULLY RESTARTED (IN CASE OF CRASH), WITH THE DATA AREA IN ITS ORIGINAL STATE, WITHOUT CATASTROPHIC FAILURE.

(4) PROCESSES USE AND/OR MODIFY FIELDS WITH PUBLICIZED NAMES - PERMITS COORDINATION.

(5) PROCESSES RETURN A 'QUIT CODE'

   a) SUCCESS
   b) FATAL ERROR
   c) TEMPORARY FAILURE

   - IN c, SUCH FAILURES INCLUDE 'FOREIGN SITE DOWN', 'DATA BASE LOCKED BY ANOTHER PROCESS', ETC.

REPRESENTATIVE PROCESSES

EXPANSION - PROCESSES 'TO' LIST, USING DATA BASE POINTED TO BY 'EXPAND-USING' FIELD, TO CONVERT NICKNAMES, GROUPS, ETC. INTO USER NAMES

TRANSMISSION - CREATES DATA AREA FOR EACH ADDRESSEE OF THE MESSAGE, AND SCHEDULES 'DELIVERY' PROCESS FOR EACH.

DELIVERY - OUTPUTS MESSAGE, USING FORMAT OBTAINED FROM VARIOUS MESSAGE FIELDS. TYPICAL OPTIONS ARE 'ON-LINE', I.E. TO A USER AT A CONSOLE, 'LOGIN-FILE', I.E. TO A FILE PRINTED WHEN A USER LOGS IN, ETC.

FILE-OUTPUT - LIKE DELIVERY, BUT OUTPUTS TO ANY NUMBER OF FILES

DELETION - DELETES THE USER'S AREA FOR THE MESSAGE, WHEN NO PENDING PROCESSES EXIST, OR LINKS INTO IT EXIST.

SPELLING - EXAMINES FIELDS SUCH AS TEXT, LOOKING FOR ERRORS.

AUTHOR-PROCESSING
RECEIVER-PROCESSING - AN ESCAPE HATCH FOR NON-STANDARD PROCESSES. APPLIES PROCESS WHICH IS CONTAINED AS A PROGRAM IN A FIELD OF THE MESSAGE. VIRTUALLY ANY PROGRAMMABLE BEHAVIOR CAN BE EFFECTED USING THIS FACILITY

ACKNOWLEDGMENT - SENDS MESSAGE BACK TO AUTHOR INDICATING WHETHER MESSAGE HAS BEEN SUCCESSFULLY DELIVERED, READ, ETC.

BACKGROUND PROCESSOR(S)

PURPOSE: TO RUN ELIGIBLE PROCESSES WHEN A USER IS ABSENT, OR UNINTERESTED IN WAITING FOR THE PROCESS(ES) TO COMPLETE.

(1) SYSTEM DATA ACCESSING PRIMITIVES MONITOR CHANGES TO ALL 'PROCESSING-NEEDED' FIELDS, AND MAINTAIN QUEUE OF ALL DATA AREAS WITH ELIGIBLE PROCESSES.
   - 'PROCESSING-NEEDED' PROVIDES FOR SCHEDULING AT A SPECIFIC TIME

(2) PROCESSOR MAY SELECT AN ELEMENT OF THE QUEUE, AND TRY TO LOCK ITS ASSOCIATED SEMAPHOR, USING PRIMITIVES PROVIDED.

   - FAILURE ⇒ ANOTHER PROCESSOR IS HANDLING THE ELEMENT, TRY ANOTHER

   - SUCCESS ⇒ RUN ELIGIBLE PROCESSES

(3) PROCESSOR HANDLES 'QUIT CODES' OF PROCESSES

   - SUCCESS ⇒ REMOVE FROM PROCESSING NEEDED

   - FATAL ERROR ⇒ REMOVE FROM QUEUE OF ELIGIBLE MESSAGES, SEND REPORT TO PERSON RESPONSIBLE FOR THE MESSAGE

   - TEMPORARY ERROR ⇒ RESCHEDULE PROCESS FOR FUTURE TRY; IF IT HAS BEEN TRIED MANY TIMES ALREADY, TREAT AS FATAL ERROR INSTEAD.

DATA BASES FOR PROCESSES

I/ 'PROCESSING-NEEDED' FIELD OF DATA AREA FOR A MESSAGE

1. NAME OF PROCESS TO BE RUN
2. EARLIEST TIME AT WHICH IT MAY RUN
3. FLAG TO DETERMINE WHETHER OTHER PROCESSES SUCCEEDING THIS ONE MUST WAIT FOR ITS COMPLETION

II/ PROCESSORS' QUEUE

1. MESSAGE # AND USER ID OF A DATA AREA WHOSE PROCESSING-NEEDED FIELD IS NON-EMPTY
2. EARLIEST TIME OF ANY PROCESS IN THE PROCESSING-NEEDED FIELD
3. NAME OF A LOCK TO BE USED WHEN HANDLING THIS QUEUE ELEMENT.

SAMPLE SEQUENCE OF EVENTS

(1) COMPOSER PROGRAM WRITES TO, TEXT, ETC INTO A FRESH MESSAGE AREA, AND ADDS "SENDING" TO THE PROCESSING-NEEDED FIELD.

(2) A BACKGROUND PROCESSOR SUCCESSFULLY LOCKS THE QUEUE ELEMENT FOR THIS MESSAGE AREA, AND BEGINS RUNNING INDICATED PROCESSES.

(3) THE 'SENDING' PROCESS IS RUN. IT ADDS 'EXPANSION', 'TRANSMISSION', FOR IMMEDIATE RUNNING, AND SCHEDULES 'DELETION' FOR 2 DAYS IN THE FUTURE.

(4) THE 'EXPANSION' PROCESS IS RUN. IT PROCESSES THE 'TO' LIST TO TRANSFORM NICKNAMES, GROUPS, ETC.

(5) THE 'TRANSMISSION' PROCESS IS RUN. IT CREATES A DATA AREA FOR EACH ADDRESSEE, ADDING 'DELIVERY' TO EACH AREAS PROCESSING-NEEDED FIELD.

(6) BECAUSE THE ONLY OTHER PROCESS (DELETION) IS NOT YET RUNNABLE, THE PROCESSOR UNLOCKS THE QUEUE ELEMENT, AND LOOKS FOR SOMETHING ELSE TO DO.

— THE FOLLOWING SEQUENCE OCCURS FOR EACH ADDRESSEE, AND MAY OCCUR AT ANY TIME AFTER (5) ABOVE. THEY MAY RUN IN PARALLEL IF SEVERAL PROCESSORS ARE AVAILABLE.

(R-1) A PROCESSOR SUCCESSFULLY LOCKS THE QUEUE ELEMENT FOR AN ADDRESSEE OF THE MESSAGE.

(R-2) THE 'DELIVERY' PROCESS RUNS, PLACING THE MESSAGE IN A READABLE FORMAT, AS DEFINED BY THE INSTRUCTIONS CONTAINED IN THE 'DELIVERY-FORMAT' FIELD OF THE MESSAGE.

(R-3) SINCE NO MORE PROCESSES ARE RUNNABLE, THE DATA AREA LOCK IS RELEASED.