Message Flow Modeling

Oscar Nierstrasz
and
Dennis Tsichritzis

Computer Systems Research Group
University of Toronto

ABSTRACT

A message management system provides users with a facility for automatically handling messages. This paper describes a technique for characterizing the behaviour of such a system in terms of message flow. Messages may be conveniently classed according to what *path* or sequences of stations they visit. Complicated or unpredictable behaviour may be modeled non-deterministically, and the resulting message paths are shown to be regular expressions.

1. Introduction

Until recently messages have been treated as a passive means of communication. Users or programs could exchange information by passing a message to an electronic mail system, which would guarantee delivery of the message to the intended recipient. Such messages typically consist of a *header* and a *body*. The message system would examine the header to determine the target and deliver the body of the message intact without interpretion. The *contents* of a message would be of no concern to the message system. In addition, a message system provides users with some editing capability and some way to file arriving messages. The requirements of an *Office Information System* [ElNu79], however, imply that a much more powerful message handling facility is needed in an office environment. Office behaviour tends to be *event-driven* [AtBD79, FiHe80, Morg80]. The arrival of a message may, for example, cause one or more procedures to be invoked. Workstations should be able to perform some of these procedures.

A *message management system* provides users with a means for automatically processing messages. Most office procedures are semi-structured [HaSi80], so such a system would enable users to describe *tasks* which follow some pattern, but may turn to the user at critical points for help. Tasks would, for example, coordinate messages, perform routine transformations (evaluate calculations or database queries), send reminders, answer queries about the messages it has seen, and draw the user's attention when unusual situations arise. Tasks may be expressed in terms of SBA boxes [deJo80], actors [Hewi77], data frames [Embl80], automatic procedures [TRGH81] or Smalltalk objects [BYTE81].

One way of incorporating these features is to assume that messages have a deep structure, not necessarily fixed, but certainly something richer than the usual amorphous body of text. Tasks may interpret messages by identifying significant ranges of numeric fields or finding patterns in text fields. The interpretation may result in a

In *Alpha Beta*, Technical Report, no. 143, pages 78-95, Computer Systems Research Group, University of Toronto, 1982.

transformation of the message, routing of the message to another station, or some local operation.

Since tasks may be triggered without user intervention, it is important to be sure that they are doing what is expected of them. An unfortunate configuration of independent tasks may result in anomalous behaviour or poor performance. Consider the case of a task that simply forwards mail for a worker on holidays. Two such tasks may exchange messages all day without anyone noticing. We may expect a sophisticated message management system to perform message flow analysis, task analysis and system instrumentation [Ruli79]. A model which supports such analysis must be capable of representing task inputs and outputs, sources and destinations, timing constraints, the amount of effort spent during each stage, *etc.* [SSKH81].

In this paper we will consider the simple case of a network of stations that communicate by sending messages. Each station may scan an incoming message, interpret it by initiating any number of actions, possibly modifying its contents, and then either discard it or forward it to some other station. It is, of course, possible to model more complicated tasks by defining substations within a station.

In order to relieve ourselves of as much detail as possible, we relate correct behaviour to flow. Messages can be separated into classes according to what sequences of stations they may visit. These sequences tell us not only what routing takes place but also the tranformations that are performed and the order of the actions that take place. The model presented provides a technique for determining what these sequences are, deciding whether or not they may terminate, and establishing what end-conditions hold if and when they do terminate.

2. The message flow model

A *message flow model* $F =< S, X, A, P >$ consists of a set $S = \{s_1, s_2, \ldots, s_m\}$ of *stations*, a set X of *messages*, and relations $A: S \times X \to X$ and $P: S \times X \to S$. A and P are the *action* and *routing* relations. They need neither be well- nor totally-defined, but they are often assumed to be functions.

A message $x \in X$ starting at some station $s \in S$ will trace a path through S by repeated applications of A and P. If we define $Q(s, x) = P(s, A(s, x))$ and $N(s, x) = (Q(s, x), A(s, x))$, then we follow the path $s, Q(s, x), Q(N(s, x)), Q(N^2(s, x)), \cdots$. We can recursively define the *path* of a message $x \in X$ starting in station $s \in S$ to be the string in S^* obtained by $\phi(s, x) = s\phi(N(s, x))$. If A and P are functions, or at least totally defined, then this path will clearly never terminate. This is also true of a message's *history*, $\Phi(s, x) = (s, x)\Phi(N(s, x))$ in $(S \times X)^*$. A history records not only the sequence of stations that a message visits, but also the contents that it had at the time.

For a given message we must designate certain stations as *initial* or *final*. To this end we augment our set S with the special stations α and ω. A station s is an *initial station* for x if $P(\alpha, x) = s$, and s is a *final station* for x if $P(s, x) = \omega$. x is created by a user or a procedure at s if s is initial, and is discarded or archived if s is final. Now A and P may be totally defined and our message paths may terminate when a message reaches its corresponding final station, by letting $\phi(\omega, x) = \Phi(\omega, x) = \varepsilon$, the empty string. Certain obvious conditions must hold: $A(\alpha, x) = x$, $A(\omega, x) = x$, $P(\omega, x) = \omega$ and $\forall s \in S\ P(s, x) \neq \alpha$.

Determining which paths a message may take is complicated by the changes of its contents. A message reaching some final station may contain contents different from the original contents. Nevertheless we attempt to approach message flow by considering *classes* of messages which may follow the same paths, rather than by solving these paths individually.

At this point, we will make the assumption that messages are of the form, at least for routing purposes, of a cross product $X = \prod_{i=1}^{n} X_i$ where each X_i is an *attribute*. Messages whose body have no structure would have a header $(X_1, X_2, \ldots, X_{n-1})$ and body X_n. We assume for the moment that messages are of fixed type with no repeating fields. Attributes which directly and indirectly affect routing are called *routing* and *control attributes* respectively. If the routing relation can be expressed in terms of only simple conditions involving the routing attributes,

and if actions set control attributes to constants after checking only simple conditions, then the collection of simple conditions provides a partition of each control attribute in a natural way [Tsic81].

If $\exists\, x, x' \in X, s \in S \cdot \supset \cdot\ x = (x_1, x_2, \ldots, x_k, \ldots, x_n)$, $x' = (x_1, x_2, \ldots, x'_k, \ldots, x_n)$, and $P(s, x) \neq P(s, x')$, then X_k is a *routing attribute*. X_k may also affect routing only after one or more applications of A and P. This is possible if X_k is used to set the value of some routing attribute. We call X_k a *control attribute* if $\exists\, s \in S, i \in \mathbf{N} \cdot \supset \cdot Q(N^i(s, x)) \neq Q(N^i(s, x'))$.

We let P_{ij} be the predicate $P(s_i, x) = s_j$. Consider such predicates of the form $P_{ij} = \vee\{C_1 \wedge C_2 \wedge \cdots C_n\}$ where each C_k is a conjunction of simple conditions $x_k < op > u$, $< op > \in \{ =, \neq, <, \leq, >, \geq \}$ and $u \in X_k$. Consider also the action A_s at station s which sets x to $A(s, x)$. If the component of A_s which modifies a control attribute X_k can be encoded as *if* $\vee\{C_1 \wedge \cdots C_n\}$ *then set* $x_k = constant$ where the C_k's are again simple conditions, then the u's of those simple conditions induce a natural partition Π_k of any control attribute X_k into ranges. If X_i is *not* a control attribute, we simply let $\Pi_i = \{X_i\}$. Note that we may choose not to model such X_i's, since they contribute no information about message flow. We thus obtain a partition $\Pi = \prod_{i=1}^{n} \Pi_i$ of X. If there are $M_i - 1$ distinct simple conditions on X_i to be found in all the A_s's and P_{ij}'s, then Π_i partitions X_i into M_i ranges. So, $|\Pi_i| = M_i$ and $|\Pi| = M = \prod_{i=1}^{n} M_i$.

We denote the *block* π_i of $\Pi = \{\pi_1, \ldots, \pi_M\}$ containing a message $x \in X$ by the equivalence class $\bar{x}$ or $[x]$. If A and P are functions as described with the above restrictions, then the extensions $A(s, \bar{x}) = [A(s, x)]$ and $P(s, \bar{x}) = P(s, x)$ are well-defined since all messages in the class $\bar{x}$ satisfy the same simple conditions on routing or control attributes.

Let $S \times \Pi$ be the set of *states* of F with respect to the partition Π. We then obtain $N: S \times \Pi \to S \times \Pi$. This means that all messages in a particular class $\bar{x}$ visit precisely the same stations, and furthermore are all mapped to the same new class by any transformation they undergo.

This is still true if the C_k's are more general, (*eg.* search for a pattern in text), but now the partitions may be as large as $2^{M_i - 1}$ since we may no longer have mutually exclusive sub-ranges. If either A or P does not satisfy the set of restrictions listed above, if they are not functions, or if an arbitrary partition Π is used, then the extensions of A, P and N are, in general, not well-defined. They can, however, always be totally-defined by adding the states $\boldsymbol{\alpha}_{\bar{x}} = (\alpha, \bar{x})$ and $\boldsymbol{\omega}_{\bar{x}} = (\omega, \bar{x})$.

Whenever we do not need to distinguish between the M α-states or ω-states, we will use $\boldsymbol{\alpha} = (\alpha, \{\bar{x}\}) = (\alpha, \Pi)$ and $\boldsymbol{\omega} = (\omega, \Pi)$. Notice that we have implicitly extended the notion of a message's *state* to include *sets* of states, *i.e.* loosely,

$$(s, \{\overline{x_i} | i \in I\}) = \{(s, \overline{x_i}) | i \in I\}.$$

The following example will be used to explain the model: a message system supports a number of stations. Two in particular are of interest. One is used by a graduate student, and the other is a completely automatic station. It maintains a repository of problems, some solved and some unsolved. The student may ask for solved or unsolved problems of varying degrees of difficulty and he (or she) may submit solutions to problems. A typical message from the student, s_1, might be

$$(GET, A\ HARD\ PROBLEM, < problem >, < solution >).$$

The < *problem* > and < *solution* > field would be blank initially, but would be filled in by s_2 and s_1 respectively.

If the student asks for a simple problem and solves it, he may ask for a new one. If he asks for a difficult or unsolved problem, then s_2, the automatic station, indicates that it would like a solution submitted by changing *GET* to *SUBMIT*. If he finds a solution, then *A HARD PROBLEM* is changed to *A SOLVED PROBLEM* (automatically, if the system can check the correctness of solutions), and the message is sent back to s_2. When s_2 receives this

message, it saves the solution to the problem and lets the message die.

The control attributes are the first two fields, since these are the only ones which affect routing. (We assume that the student is smart and *always* finds a solution to any problem, so < *problem* > and < *solution* > do not affect the flow).

For our purposes, then,

$X = X_1 \times X_2$,

$\Pi_1 = \{\{GET, RETRIEVE, \cdots\}, \{SAVE, STORE, \cdots\}\}$

$\Pi_2 = \{\{SOLVED\ PROBLEM, \cdots\}, \{HARD\ PROBLEM, \cdots\}\}$.

A message can therefore be in eight states depending on which of the four classes it is in and which station it is at. There are two more states, $\boldsymbol{\alpha}$ and $\boldsymbol{\omega}$, corresponding to message creation at s_1 and message termination at s_2.

2.1. Bit vectors

Whenever the partition Π of X is of the form $\Pi = \prod \Pi_i$ where Π_i partitions X_i, we can express certain collections of message classes by *bit vectors*. Consider a message $x = (x_1, x_2, \ldots, x_n)$. Each $\Pi_i = \{\pi_{i1}, \ldots, \pi_{iM_i}\}$. Consequently each x_i belongs to some $\pi_{ij_i} \in \Pi_i$. We therefore have $\bar{x} = \prod_{i=1}^{n} \pi_{ij_i} \in \Pi$. The *bit vector* $\mathbf{b} = (b_1, \ldots, b_n)$ representing $\bar{x}$ is obtained by setting $b_i = 2^{j_i - 1}$.

For example, if $x = (GET, A\ HARD\ PROBLEM, ,)$, then $\bar{x} = \pi_{11} \times \pi_{22}$ and $\mathbf{b} = (1, 2)$.

If we wish to consider x_i belonging to any of *several* π_{ij}'s, then we *add* their respective b_i's. In particular, if we want to allow $x_i \in \pi_{ij}$ for any $j \in J_i \subseteq \{1, 2, \ldots, M_i\}$ then $b_i = \sum_{j \in J_i} 2^{j-1}$. b_i therefore ranges from 0 to $2^{M_i} - 1$. If each b_i is written in base 2, so that $\mathbf{b}$ is a vector of strings over $\{0, 1\}$, then we can easily "read" the represented blocks of Π. In the above example, $\mathbf{b} = (01, 10)$. $\mathbf{b} = (10, 11)$ would represent $(10, 10) \bigcup (10, 01)$, and so on. The "on" bits stand for π_{ij}'s: if the jth bit (reading from the *left*) of b_i is on, then we are allowing $x_i \in \pi_{ij}$. The total set of "allowable" message classes $\bar{x} \in \Pi$ is the *cross product* of the π_{ij}'s represented by the bit strings, *i.e.* $\prod_{i=1}^{n} \{\pi_{ij} | j \in J_i\}$.

Note that if each $b_i = 2^{M_i - 1}$, *eg.* $\mathbf{b} = (11, 11)$, then all the bits are on, and $\bar{x}$ is allowed to range over all of Π. Conversely, if any $b_i = 0$, *eg.* $\mathbf{b} = (10, 00)$, *then the cross product is empty, and no* $\bar{x} \in \Pi$ is represented.

A message flow model F under a partition $\Pi = \prod_{i=1}^{n} \Pi_i$ may be represented by a directed graph $D = (V, E)$ where $V = S \bigcup \{\alpha, \omega\}$ and a (directed) edge (s_i, s_j) exists for every non-empty P_{ij}. Every edge is labelled by a set of bit vectors representing all the message classes which are allowed to flow along that arc. If the P_{ij}'s consist solely of simple conditions, then a single bit vector will often suffice to label each arc. We also label each vertex by those bit vectors representing messages that may undergo a change of class. Furthermore, the change of class is explicitly indicated by marking with an asterisk the bit in b_i corresponding to the new block in Π_i of the image of x_i in $A(s, x)$. For example, s_1 only submits *SOLVED* problems. Mapping *SUBMIT* messages into *SUBMIT SOLVED* messages is encoded by $(10, 11^*)$, *i.e.* message classes $(10, 10)$ and $(10, 01)$ are mapped into $(10, 01)$. We call the labelled graph a *message flow graph* (see figure 1). Note that not every bit string in $\mathbf{b}$ need be marked.

To determine what paths are followed by all message classes, we "push" the bit vector representing all of Π through the message flow graph. When the bit vector encounters an action, it "splits" into the component affected and the one which is unaffected. Furthermore, the affected bit vector acquires the marking of the action. A marked bit vector is remarked upon encountering an action. The bit strings marked in the action are those remarked in the bit vector of the message. In this way we maintain both the original message class and the *current* message class. For the purpose of applying the action and routing relations, only the current class (or set of classes) is relevant.

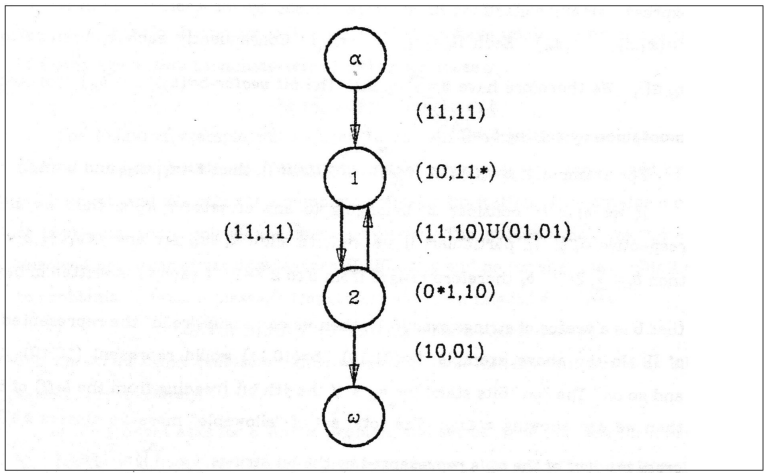

Figure 1: *A message flow graph*

Suppose, for example, that the set of messages represented by the bit vector $\mathbf{b} = (11^*, 11)$ encounters the action represented by $(0^*1, 10)$. **b** started its life as $(11, 11)$ but is currently $(01, 11)$. Only the current value is of importance in evaluating the effect of the action, but we must also keep track of the original value. The action causes **b** to be split into $(11^*, 10)$ and $(11^*, 01)$, *i.e.* the component affected by the action and the one unaffected. Applying the action causes the bit vectors to be remarked, and we obtain $(1^*1, 10)$ and $(11^*, 01)$.

Bit vectors are also split according to routing. When the (current) message classes represented by a bit vector must follow different arcs, that bit vector must be split into the appropriate components. There may be several ways to do this: Suppose $(11, 11)$ is routed in two directions. If one of the arcs is labelled $(10, 01)$ and all other message classes follow the other arc, then the second component of $(11, 11)$ can be represented by $(11, 10) \bigcup (01, 01)$ or $(10, 10) \bigcup (01, 11)$ or $(10, 10) \bigcup (01, 10) \bigcup (01, 01)$. For illustration see figure 1 and for a more detailed account see [Tsic81].

2.2. Non-determinism in the MFM

Although A and P are defined as relations, we have, up till now, more or less assumed them to be functions. We have also placed some ostensibly unreasonable restrictions on the nature of the actions and routing predicates. It is unlikely, in practice, to only have actions that set attributes to constants. It is also unreasonable to expect that simple conditions will suffice to express all desired routing predicates. Finally, it is possible that A and P depend on some external information (such as user input), thus rendering them non-deterministic within the framework of the message flow model.

It is clear that this case prevents A or P from being well-defined. Furthermore, if actions do not set attributes to constants, or if arbitrary conditions are used in routing predicates, then A and P may be well-defined over $S \times X$, but not always over $S \times \Pi$. An arbitrary action may map one message class into any number of other message classes. $E_{S\times\Pi}$ induced by the partition $S \times \Pi$, is necessarily an equivalence relation, but may not be a congruence relation for N. By the same token, arbitrary routing functions applied to a state $(s, \bar{x})$ may not map neatly to a single station for each $x \in \bar{x}$. It may be possible to choose Π so that A and P are well-defined over states, but, in the worst case, $\Pi = \{\{x\} | x \in X\}$. We must therefore be prepared to handle non-determinism if we expect to obtain any worthwhile reduction of message classes in some situations.

A non-deterministic message flow graph differs from its deterministic counterpart only in that the bit vectors labelling out-edges from a station (*i.e.* routing predicates) may not be mutually exclusive, and that the bit-vectors labelling stations (*i.e.* actions) may have several marked bits in a given bit string. The multiple marking of an

action's bit vector indicates the *image* of that action when applied to messages represented by the bit vector.

The non-determinism introduced into the model now prevents each message class from being associated with a particular path. The collection of possible message paths taken by messages in a block of Π can, however, be expressed by *regular expressions* [Ginz68]. We first extend the notion of a message's *history* $\Phi(s, x)$ over $(S \times X)^*$ to message states. We have, therefore, $\Phi(s, \bar{x}) = (s, \bar{x})\Phi(N(s, \bar{x}))$ yielding a string or a choice of strings in $(S \times \Pi)^*$.

Theorem 1: Message histories in a non-deterministic message flow model $F =< S, X, A, P >$ can be expressed by regular expressions.

Proof: Let Π be a partition of X. Consider the finite automaton $A =<V', V', \xi, \boldsymbol{\alpha}, \{\boldsymbol{\omega}\} >$. The *states* V' of the automaton are the states of the message flow model, $S \times \Pi \bigcup \{\boldsymbol{\alpha}, \boldsymbol{\omega}\}$, together with the special α- and ω-states The *inputs* are also V'. The *next-state* relation $\xi(v_1, v_2)$ holds

(i) if $v_1 = \boldsymbol{\alpha}$ and v_2 is initial
(ii) if $N(v_1) = v_2$
(iii) if v_1 is final and $v_2 = \boldsymbol{\omega}$.

Note that $N(s_1, \bar{x}_1) = (s_2, \bar{x}_2)$ if$\exists\, x_1 \in \bar{x}_1,\ x_2 \in \bar{x}_2 \cdot\supset\cdot N(s_1, x_1) = (s_2, x_2)$. Neither N nor ξ need be well-defined. Since the input strings accepted by A are precisely the valid histories in F of message classes in Π, those histories may be expressed by regular expressions □.

Corollary 2: Message paths in F may be expressed by regular expressions.

Proof: The message paths are obtained from the regular expressions in *theorem 1* by the mapping $v_1(s, \bar{x}) = s$ extended to strings in $(S \times \Pi)^*$. Alternatively, consider the automaton $A' =< V', S', \xi', \boldsymbol{\alpha}, \boldsymbol{\omega} >$, where $\xi'(v_1, s_2)$ holds if $\exists\, v_2 = (s_2, \bar{x}_2) \cdot\supset\cdot \xi(v_1, v_2)$ holds. The strings accepted by A′ are the valid message paths taken by message classes in Π, and are therefore regular expressions □.

The *transition graph* [Ginz68] of A is the directed graph $D'(V', E')$ where $E' = \{(v_1, v_2) | \xi(v_1, v_2)\}$. D' may be thought of as an inefficient representation of the message flow graph, D. We call D' the *state-graph* of F. (See figure 2.)

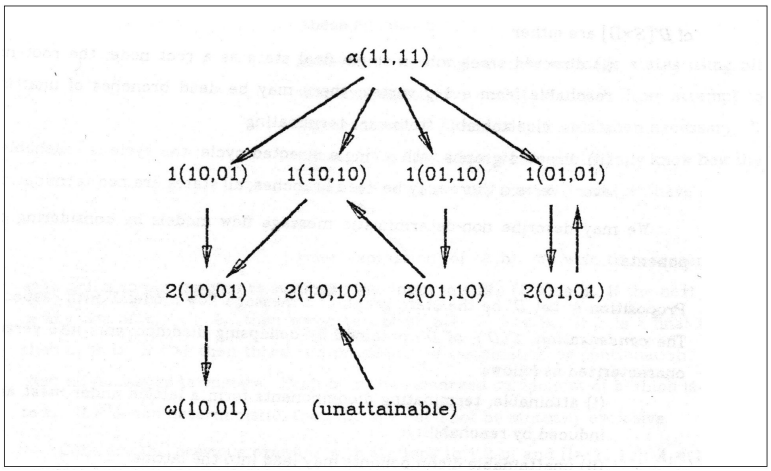

Figure 2: *A state graph*

Any state not reachable from $\boldsymbol{\alpha}$ is *unattainable*. Any state from which $\boldsymbol{\omega}$ is reachable is *terminating*. If D' has no unattainable states, then it has exactly one vertex $\boldsymbol{\alpha}$ with zero indegree and one vertex $\boldsymbol{\omega}$ with zero outdegree. If F is a deterministic message flow model with respect to Π, then every vertex $v \in V' \setminus \{\boldsymbol{\alpha}, \boldsymbol{\omega}\} = S \times \Pi$ has outdegree $\delta^+(v) = 1$.

Two vertices in a directed graph are *diconnected* if each is reachable from the other. Diconnection is an equivalence relation whose resultant partition induces subdigraphs called *dicomponents* [BoMu76]. If F is deterministic, then the non-trivial dicomponents of D' are cycles: if a message encounters a state twice in its history, it must, *ipso facto*, cycle indefinitely. (Not so if F is non-deterministic.) We have, therefore, the following classification:

Proposition 3: If a message flow model F is deterministic with respect to Π, then the history of an initial message class is either $v_{i_1} v_{i_2} \cdots v_{i_k}$ or

$$v_{i_1} v_{i_2} \cdots v_{i_l} (v_{i_{l+1}} \cdots v_{i_k})^{\infty}.$$

Furthermore, the state graph D' of F has the property that the connected subgraphs of $D'[S \times \Pi]$ are either

(i) directed trees with a single final state as a root node; the root node is reachable from every vertex; there may be dead branches of unattainable states; all attainable states are terminating

(ii) directed graphs with a single directed cycle; the cycle is reachable from every vertex; there may be dead branches; all states are non-terminating □.

We may describe non-deterministic message flow models by considering dicomponents:

Proposition 4: Let D' be the state graph of a message flow model F with respect to Π. The *condensation*, $C(D')$, of D', obtained by collapsing dicomponents into vertices, is characterized as follows:

(i) attainable, terminating dicomponents form a lattice under *meet* and *join* induced by reachability

(ii) unattainable dicomponents may lead into the lattice

(iii) *critical* dicomponents may lead out of the lattice to non-terminating dicomponents □.

One immediate consequence of this is that once a message has left a dicomponent it may never return to it. Since a condensation, or dicomponent-graph, is acyclic, a message may never return to a state once it has entered a new dicomponent. This may lead us to regular expression for message paths which are simply the concatenations of the regular expressions for dicomponents.

We may use classical techniques for deriving the regular expressions, but we may save some effort by using D, the message flow graph, instead of D', the state-graph, as our representation of F and of our finite automaton. Regular expressions are obtained by solving $T = BT + E$ where T and E are column vectors and B is a matrix. Each T_i is the regular expression we seek for state i of the automaton. E_i is $\wedge$ (the empty string) if i is a final state and $\varnothing$ otherwise. The entries B_{ij} are the inputs required to advance from state i to state j. The solution to T is $B^* E$, where $B^* = I + B + B^2 + B^3 \cdots$, but this format is useless to us for determining the individual T_i's. Regular expression are non-commutative, so conventional matrix equation solving techniques are also useless here. (They *are* useful, however, for solving the generating functions for each regular expression, since those do commute.) The laborious method of substitution and elimination is therefore used to solve these equations.

We may save some effort, however, if we conglomerate message states using bit vectors. Instead of deriving regular expressions for each $(s, \bar{x})$ in $S \times \Pi$, we attempt to do so for $(s, \mathbf{b})$ where $\mathbf{b}$ initially represents all of Π, and is split only when necessary. It is not possible to write a matrix equation for this, since we do not initially know how the bit vectors will be split. We therefore do not know the size of T until we have already solved it.

We let $k\mathbf{b}$ represent the regular expression for $(k, \mathbf{b})$. We wish to represent message paths so our inputs are stations. Our input in state $(k, \mathbf{b})$ is k. If the next station is any one of $k_1, \ldots, k_t$, then we write $k\mathbf{b} = kk_1\mathbf{b}_1 + \cdots + kk_t\mathbf{b}_t$. If k is a final station, then k_t is ω. If $t > 1$ then there is a possibility of termination or continuation. If $t = 1$, then all messages terminate. Each $\mathbf{b}_i$ is the *remarked* component of $\mathbf{b}$ which is routed to k_i. If F is non-deterministic, then the $\mathbf{b}_i$'s need not be mutually exclusive.

Consider the previous example with stations $\{\alpha, 1, 2, \omega\}$ and $\Pi = (11, 11)$: $A_1 = (10, 11^*)$, $A_2 = (0^*1, 10)$, $P_{\alpha 1} = (11, 11)$, $P_{12} = (11, 11)$, $P_{21} = (11, 10) \cup (01, 01)$ and $P_{2\omega} = (10, 01)$. Messages enter the system at station 1,

get passed to station 2 and back, and possibly leave the system at station 2. Messages may be modified at either station depending on their contents.

We write:

$$\alpha(11,11) = \alpha 1(11,11)$$
$$1(11,11) = 1(10,11) + 1(01,11)$$
$$= 12(10,11^*) + 12(01,11)$$
$$2(10,01) = 2\omega(10,01)$$
$$2(01,11) = 2(01,10) + 2(01,01)$$
$$= 21(0^*1,10) + 21(01,01)$$
$$1(10,10) = 12(10,10^*)$$
$$= 12\omega(10,10^{*)}$$
$$1(01,01) = 12(01,01)$$
$$= (12)^\infty(01,01)$$

Substituting back we obtain:

$$\alpha(11,11) = \alpha 12\omega(10,11^*)$$
$$+\alpha 1212\omega(0^*1,10^*)$$
$$+\alpha(12)^\infty(01,01).$$

We may now "read" the message paths as being the strings preceding the message classes in the above expression. In this example there are two terminating message paths. All terminating messages end in the class $(10,01)$. Only messages in the class $(01,01)$ do not terminate. (Note that $(2,(10,10))$ is an unattainable state and, as such, $2(10,10)$ appears nowhere in the above equations. $2(10,11^*)$ is really $2(10,01)$ since we are interested only in *current* states.) The only non-trivial dicomponent is $\{1(01,01), 2(01,01)\}$. A terminating dicomponent would make a contribution to the regular expression of the form $(expression)^*$. We have saved a small amount of work by using bit vectors, since we would have otherwise used 16 equations for 4 stations times 4 message classes.

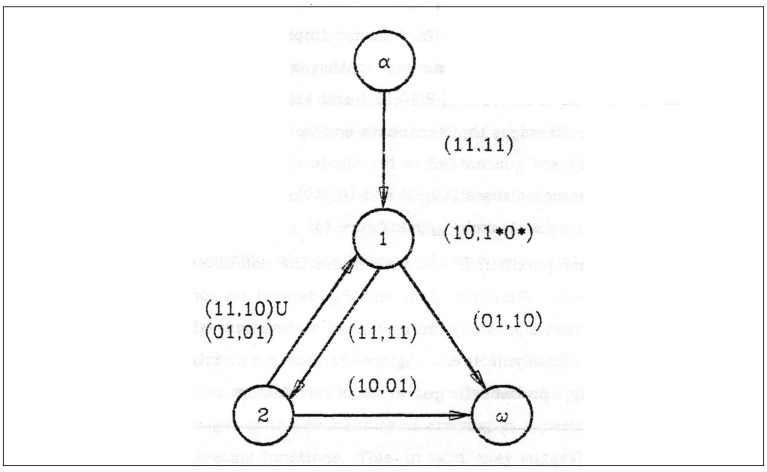

Figure 3: *A non-deterministic message flow graph*

The next example illustrates how non-determinism is handled by the model: we have stations $\{\alpha, 1, 2, \omega\}$ and $\Pi = (11,11)$ as before. Furthermore, $A_1 = (10, 1^*0^*)$, $A_2 = (11,11)$, $P_{\alpha 1} = (11,11)$, $P_{12} = (11,11)$, $P_{1\omega} = (01,10)$, $P_{21} = (11,10) \cup (01,01)$ and $P_{2\omega} = (10,01)$ (see figure 3). We have introduced two kinds of non-determinism: A_1 either depends on some external parameter, or our partition Π is not fine enough to give us congruence classes over X_2. In our example the student is allowed to submit "solutions" to problems which he decides to classify as *SOLVED* or *UNSOLVED*. In the latter case he may be submitting a tentative solution and asking for a hint. We

have a similar situation with P_{12} and $P_{1\omega}$. Since they are not disjoint, our routing is non-deterministic: The student now has a *choice* of requesting a fresh *HARD PROBLEM* or giving up. s_2 no longer insists on getting a solution. Nevertheless we may proceed as before:

$$\begin{aligned}
\alpha(11,11) &= \alpha 1(11,11)\\
1(11,11) &= 1(10,10) + 1(01,10) + 1(11,01)\\
&= 12(10, 1^*0^*) + 1\omega(01,10) + 12(01,10) + 12(11,01)\\
2(10,11) &= 21(10,10) + 2\omega(10,01)\\
&= 212(10, 1^*0^*) + 2\omega(10,01)\\
&= 2(12)^*\omega(10, 11^*)\\
2(01,10) &= 21(01,10)\\
&= 212(01,10) + 21\omega(01,10)\\
&= 21(21)^*\omega(01,10)\\
2(11,01) &= 2\omega(10,01) + 21(01,01)\\
1(01,01) &= 12(01,01)\\
&= (12)^\infty(01,01)
\end{aligned}$$

Substituting back we obtain:

$$\begin{aligned}
\alpha(11,11) = {} & \alpha 12(12)^*\omega(10,10^*)\\
& + \alpha 1\omega(01,10)\\
& + \alpha 121(21)^*\omega(01,10)\\
& + \alpha 12\omega(10,01)\\
& + \alpha(12)^\infty(01,01) \quad .
\end{aligned}$$

Notice that the second and third terms simplify to $\alpha 1(21)^*\omega(01,10)$. As before we can read off the terminating message paths: $\alpha 12(12)^*\omega$, $\alpha 1(21)^*\omega$ and $\alpha 12\omega$. The new dicomponents are $\{1(10,10), 2(10,10)\}$ and $\{1(01,10), 2(01,10)\}$. There are no unattainable states. Messages that terminate end up in classes $(10,01)$ or $(01,10)$. Messages in class $(10,01)$ are guaranteed to terminate. Messages in class $(01,01)$ will never terminate. Message classes $(10,10)$ and $(01,10)$ may or may not terminate depending on the true nature of A_1 and $P_{1\omega}$ (see figure 4).

If a finer partition Π would eliminate the non-determinism then those paths will also terminate. Similarly, if A_1 and $P_{1\omega}$ depend on some external variable, then we most show that a provably finite number of iterations will enable our messages to exit their loops. Finally, if A_1 and $P_{1\omega}$ are "genuinely non-deterministic" (*i.e.* random), then we have only a probablistic guarantee of termination.

3. Concluding remarks

The message flow model presented in this paper provides us with a way to characterize tasks in a message management system. The regular expressions which describe the paths taken by classes of messages may be used in the analysis of a given system for correct behaviour. We will continue to extend and develop the model. Coordination of messages, for example, is not yet represented. The power of hierarchical decomposition of stations into substations corresponding to tasks and subtasks has not been explored. We also hope to incorporate the model within a working message management system capable of specification of automatic message handling procedures [TRGH81].

The area of regular expressions also suggests many possible applications of the model, in particular, the assignment of weights to stations will yield generation functions for costs associated with classes of messages. The amount of work taken to process a particular message, or the probability of arriving at a particular state may be determined using generating functions. This, in turn, may suggest ways of improving performance by redistributing workloads or restructuring the system.

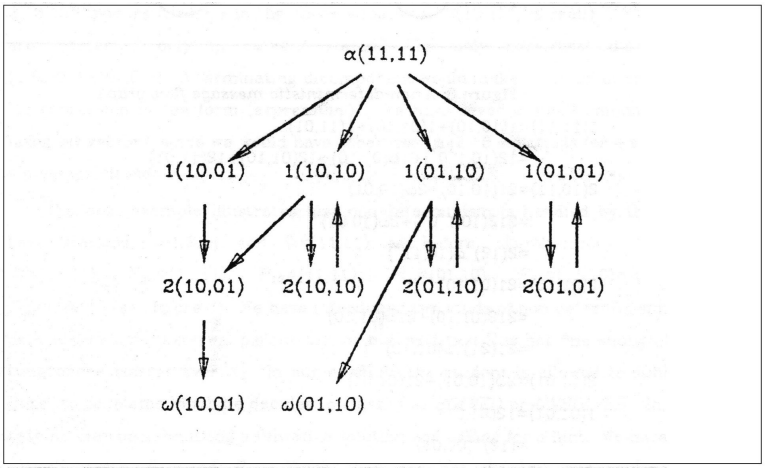

Figure 4: *A non-deterministic state graph*

4. Bibliography

[AtBS79] Attardi, G., Barber, G. and Simi, M., "Towards an Integrated Office Work Station", AI Laboratory, MIT, Cambridge, 1979.

[BoMu76] Bondy, J.A. and Murty, U.S.R., *Graph Theory with Applications*, North Holland, New York, 1976.

[BYTE81] Special issue on Smalltalk, Byte Vol. 6, No. 8, Aug. 1981.

[Cook79] Cook, C.L., "Streamlining Office Procedures", System Sciences Lab, Office Research Group, SSL79-10, Xerox PARC, Nov. 1979.

[deJo80] deJong, P., "The System for Business Automation: A Unified Application Development System", IBM Research Report #34539, Yorktown Heights, N.Y., 1980.

[deZl77] deJong, P., Zloof, M.M., "Communications within the System for Business Automation", IBM Research Report #28537, Yorktown Heights, N.Y., July 1977.

[Eile74] Eilenberg, S., *Automata, Languages and Machines*, Volume A, Academic Press, New York, 1974.

[Elli79] Ellis, C.A., "Information Control Nets", Proceedings of the ACM, Conference on Simulation, Measurement and Modification, Boulder, Colorado, Aug. 1979.

[ElNu79] Ellis, C.A. and Nutt, G.J., "Computer Science and Office Information Systems", Xerox PARC, June 1979.

[Embl80] Embley, D.W., *A Forms-based Nonprocedural Programming System*, Department of Computer Science, University of Nebraska-Lincoln, 1980.

[FiHe80] Fikes, R.E. and Henderson, D.A., "On Supporting the Use of Procedures in Office Work".

[FoFu62] Ford, L.R. and Fulkerson, D.R., *Flows in Networks*, Princeton University Press, Princeton, 1962.

[GaJo79] Garey, M.R., and Johnson, D.S., *Computers and Intractability: A Guide to the Theory of NP-completeness*, Freeman, San Francisco, 1979.

[Ginz68] Ginzberg, A., *Algebraic Theory of Automata*, Academic Press, New York, 1968.

[HaKu80] Hammer, M. and Kunin, J.S., "Design Principles of an Office Specification Language", NCC 1980.

[HaPa73] Harary, F. and Palmer, E.M., *Graphical Enumeration*, Academic Press, New York, 1973.

[HaSi80] Hammer, M. and Sirbu, M., "What is Office Automation?", Office Automation Conference, Georgia, 1980.

[Hewi77] Hewitt, C., "Viewing Control Structures as Patterns of Passing Messages", Artificial Intelligence, Vol. 8, No. 3, pp. 323-364, June 1977.

[HoNT81] Hogg, J., Nierstrasz, O.M. and Tsichritzis, D., "Form Procedures", to appear in IEEE Transactions on Software Engineering.

[LaTs80] Ladd, I. and Tsichritzis, D., "An Office Form Model", Proceedings NCC, Anaheim, 1980.

[LuYa81] Luo, D. and Yao, S.B., "Form Operation By Example", Department of Computer Science, Purdue University, W. Lafayette, Indiana, 1981.

[Macq79] MacQueen, D.B., "Models for Distributed Computing", INRIA Research Report #351, Domaine de Voluceau, Rocquencourt, Le Chesnay, France, April 1979.

[Morg80] Morgan, H.L., "Research and Practice in Office Automation", Wharton School, University of Pennsylvania, Philadelphia, 1980.

[Pete77] Peterson, J., *Petri Nets*, ACM Computing Surveys Vol. 9, No. 3, pp. 223-252, Sept. 1977.

[Pott78] Potts, D., *Specifications Language for Office Procedures Execution*, Thesis, The Wharton School, University of Pennsylvania, Philadelphia, 1978.

[Ruli79] Rulifson, J.F., "Information Systems Management and Misapplied Methodologies", Inter-office memo, XEROX PARC, March 21, 1979.

[SSKH81] Sirbu, M., Schoichet, S., Kunin, J. and Hammer, M., "OAM: An Office Analysis Methodology", MIT Office Automation Group Memo OAM-16, Cambridge, 1981.

[TRGH81] Tsichritzis, D., Rabitti, F.,Gibbs, S., Hogg, J., Nierstrasz, O., and Kornatowski, J., "A Message Management System", IEEE Transactions on Communications, Vol. 30, No. 1, January 1982.

[Tsic81] Tsichritzis, D., "Form Management" to appear in CACM and in *Omega Alpha*, Technical Report 127, Computer Systems Research Group, University of Toronto, 1981.

[Zism76] Zisman, M.D., "A Representation for Office Processes", working paper, Wharton School, University of Pennsylvania, Philadelphia, 1976.

[Zism77] Zisman, M.D., "Representation, Specification and Automation of Office Procedures, PhD thesis, Wharton School, University of Pennsylvania, Philadelphia, 1977.