ANALYSIS AND DECOMPOSITION OF COMPLEX REAL-TIME PROCESSES WITH A REAL-TIME SCHEMATA MODEL[1]

W. R. ADRION, P. A. FRICK and P. A. SZULEWSKI
Oregon State University
Corvallis, Oregon

Summary. In this paper a formal model for real-time processes is expanded to analyze structured, complex algorithms. Structured forms have inherent modularity and the authors discuss advantages of mapping such processes on distributed micro or mini processor networks. Processes under consideration are subject to random interrupts by independent I/O demands. Degradation of total system performance due to these demands is discussed. Extentions and ongoing research are mentioned.

Introduction. The authors introduce a structured hierarchical graph model designed to identify and analyze information production, flow and loss in the implementation of a real-time process in a digital computer. The model first given in [A1][A2] combines some basic concepts of information with a structured modular decomposition of a complex real-time process to produce criteria useful for real-time system performance evaluation.

Real-time data processing has many connotations, however, for the purpose of this paper, real-time processing is concerned with the acquisition and sensing of externally generated data, not readily reproducible, in a time constrained domain and can exercise a time critical control action producing a physical result.

Information that a real-time process deals with differs from the strictly quantitative definition of information an information theorist might propose. Information in real-time is processed, according to a rule specified by a well defined sequence of operations, not merely transmitted through a channel. As a result, it has a qualitative as well as a quantitative aspect. Information quality is a measure proposed as a function of both the reliability of a process to reach its objective and the processing time available to the process.

Real-Time Software. Software requirements for systems designed for real-time data acquisition and control vary according to the needs of the particular user, however, they

[1] This work was supported by NSF under grant number ENG73-03645

can be categorized in terms of their function within the system. Real-time software will be one of three forms.

a. Applications software: a program in this category would generally perform a specific task. Examples in this category include solving equations, missile tracking, controlling a machine tool, or monitoring processes.

b. Software to aid development of applications software: this category includes assemblers, higher level language compilers, interpreters, and debugging tools.

c. Control or system software: the operating system, monitor or executive programs would comprise this category.

This paper is essentially concerned with the analysis of applications software as implemented in a real-time environment, implementations of processes that are subject to external as well as internal interrupts.

Interrupts . There are two distinct classes of interrupts that can be related to rea1-time data processing the class known as traps include all those faults, that cause an abnormal exit, from the currently executing process, to another process initiated to check the fault. Conditions that can cause such an interrupt include all those unpredictable events that are specific to the running process; addressing faults, protection violations, arithmetic overflows and interval time outs. External interrupts, those used for system input or output and peripheral device status, are generated by external sources or affect an external source. External interrupts can arise as a result of the processor being used to control a number of separate, but not necessarily mutually exclusive tasks. They are treated as high priority messages to or from external sources and by this relation to the processing can be considered harmful through their ability to degrade total system performance.

Since the probability of multiple interrupts is high there exists an interrupt scheduler, typically a hardware device, that identifies and initiates a process according to some queue discipline when an interrupt occurs. The run time priority can be a static or dynamic assignment. The interrupt handler makes the processor act as a part time hardware message queuing facility.

The queue discipline used to assign process order can take a number of forms. The most common is first-in, first-out abbreviated FIFO. Other common queue disciplines are last-in, first-out or LIFO, Service in Random Order, SIRO, and priority service, PRI.

Most real-time processes operate under priority service queue disciplines. Each process is assigned a priority class number say from 1 to n, where the lower priority class number has higher process priority. Processes within the same priority class are served in order of arrival (FIFO). In situations where a process with priority class i interrupts a process with priority class j and i>j, preemptive or non-preemptive priority control must be defined.

Structured Modular Design. To reduce program size and complexity to levels of human managability, approaches have been designed around the concept of modularity. One of the principle techniques is a process of stepwise decomposition of the problem with simultaneous stepwise composition developed by Dijkstra [D1]. By dividing a task into subtasks in this manner, the correctness or validity of the overall structure is dependent on the correctness of the components.

Dijkstra's approach to the development of a structured program is implemented in a Top-Down manner. A general statement of the total problem supplies the top level description and in successive lower levels, each previous level is refined by division into substructures, concentrating on resolving critical points at that level rather than dwelling on issues that can be put off until lower levels. At each level, the structure should still exhibit a correct and complete form of the original problem statement. When the refinement process is complete, each sub-structure is coded in the machine dependent target language.

In decomposing a process into modules some structured programming constructs and input-output relationships have to be considered. Structured Programs exhibit a form characterized by go-to less structural elements. To say there are no go-to's is an overstatement, since at some conceptual level branching backward or foreward is a necessary structural element. The if-then-else construct (Figure 3b) and the Do-While (Figure 3c) are looping structures that have painfully eliminated the obvious go-to's. These constructs, along with a sequential one-step construct (Figure 3a) make up forms from which complex processes are composed.

Real-Time Process Description. In order to maintain enough generality in discussing processes not encoded in a particular programming language, the concept of uninterpreted operations is used as the basic units of a programming structure. Operations can be broadly classified as: 1) conditional - some criterion selects the next sequential instruction from two alternatives; 2) unconditional - a sequential function is employed and passes control to the next operation. A conditional instruction has a branch successor that is enabled when a condition is met, and a fall through successor that is enabled when the condition is not met. Not interpreting operations allows the model flexibility in its ability to describe a large class of operations one might encounter, from micro instructions to high level language functions.

Any sequence of legal operations that is structurally well defined can be considered a process. Figure 7a represents a sample structured process in flow chart form. The representation of the process shows the spatial relationships between each operation and its successor.

An analytic and descriptive formulation of the process, that retains the branching and looping substructure characteristics, can be generated as a regular expression of operations. The process described by Figure 7a is, for example, equivalent to the formula:

$$P = a_1[a_2a_3a_4a_5a_6(a_7+a_9a_{10})a_8]^*a_2a_3 \tag{1}$$

This provides both a path-wise and a module-wise decription of a structured process. Branching and looping substructures are obvious by inspection of equation (1).

The operator + and implicit · are defined as logical union and disjunction respectively. The operator * (iterate) over an operation or a closed set of operations implies that the operation or operation set is to be executed zero or more times. It is not uncommon to specify a fixed number n rather than * to indicate a fixed number of iterations. To further simplify the representation, the operations can be symbolized in terms of the groupings defined previously corresponding to fall through and branch successor relationships. A conditional operation a_i with its branch successor a_j and fall through successor a_k will be reduced by replacing a_i (a_j+a_k) with $a_i + a_i'$ where a_i is the fall through operation and a_i', the branch operation. This replacement will yield the regular expression notation as:

$$P = a_1[a_2a_3a_4(a_5+a_5'a_6)a_7]^*a_2a_3' \tag{2}$$

with the flow graph representation in Figure 7b. Note that in the flow graph, the primed operation is implicit and occurs when the branch successor option i enabled.

Subpaths in the graph can be determined by inspection of the regular expression formula, with the added feature, that specifies fixed or indefinite looping sub-structures, the * operator.

With these features a regular expression desciptor of a process provides at least two levels of analysis. An a-posteriori path analysis of any structured process can be made, this can yield path length estimates, useful to determine execution time of a sequence of operations, and a structural analysis can identify modules within a complex process and exhibit relationships and dependencies between modules.

Complex Real-Time Processes. Structured real-time processes can be composed into modules by the scheme outlined in the preceding section. Computations performed, corresponding to the process, are concatenations of two forms of program modules one step and iterative. All processes can be constructed from these forms. A one-step process

is one which reaches a terminal state in a fixed time interval, excluding the time used to handle interrupts. Processes of this class may not necessarily consist of one operation but being a structured module can be collapsed to one-step with a fixed execution time. A one step process can be viewed mathematically as a mapping on the memory from the memory contents just prior to the initiation of the process to the memory contents at the logical completion of the process via the functional operations of the process. In contrast to the one-step process there is the iterative process. This class again may be viewed as a mathematical mapping with identical domain and range requirements. The difference between this and the one-step is that the function, as represented by the program module, may not terminate in a fixed time interval. However, the process may reach a logical point, in some fixed time, where the domain may be sampled for an intermediate result. Further iterations of the process may or may not refine the answer and the logical termination of the process is subject to the imposition of a stopping criterion. The invocation of these criteria is rather arbitrary and may or may not be mapping dependent.

Processes in the one-step class are relatively simple to analyze. Execution times, related to paths through the process, are well defined. Coupled with the execution time of a process is the reliability of the information a process produces. Deviation from the ideal output constitutes an information loss; as for example 1) quantization loss from a continuous input space to a discrete space, 2) roundoff or truncation of data words due to word length, 3) a computation becoming input, output or compute bound and, 4) premature termination of a process to satisfy a time constraint.

Processes in the iterative class are more complex. Consider for example a process of approximating an ideal mapping from a continuous space onto continuous space. The reliability of the discrete approximation depends on two factors: i) “goodness” of the approximation; ii) time of convergence. This convergence, in a one-step process, is fixed by the execution time, while in the iterative case it can be a function of the qualitative criteria and varies according to the parameters of the process or any other imposed stopping criterion.

Topological Considerations. An appropriate mathematical topology to study real-time process maps has not been developed largely because of measure theoretic problems. The foundation of a topology is based on a continuous function f that is one-to-one on a problem space S such that the function f^{-1} exists. Finite memory models with fixed size word length normally yield many to one maps. In sampling the problem space an arbitrary choice of a sampling interval cannot be made since each point (each memory element) is an isolated point in E. Measures on computed numbers, need not be homogeneous or additive in any sense, e.g. addition of two 16 bit numbers that cause overflow would yield a number not representable in 16 bits.

In order to develop a useful model for a real time system some intuitive notion of the system configuration is necessary. One can take a very general approach and represent the system as an information processing device communicating with the real world via transducers. A more realistic depiction of the behavior of a real time system would have each subsystem represented by a mapping from one mathematical space into another. Elements of a continuous (input) space are mapped under the composition of two functions, f_1 which discretizes the information and f_2 which encodes the information, into a portion of a discrete space. This discrete space represents the computer memory. An algorithm is then performed, which could be a single step or multistep iterative or recursive procedure. The algorithm can be represented as a single or series of 1-1 and onto mappings of the discrete memory space back into itself. Finally the processed information is returned and represented by the compositional mapping f_k,f_{k+1} which takes elements of the memory space into the continuous output space. One advantage of such a model is that it can be compared with the ideal mathematical function being implemented by the digital subsystem.

Real-Time Schemata. In this section the Real-Time Schemata is defined as a structured hierarchical graph model of a real-time process. The basic structure of the model was first proposed in [A1] and [A2] and the description here includes modifications necessary to analyze larger and more complex processes.

Ianov [Y1] proposed a general schema model which provides adequate analysis of computations where simple sequential control is assumed. Ianov's schemata formulation was modified by R. M. Karp and R. E. Miller [K1] to include parallel computations. Both of these schemata do not model the random interrupt in a computational sequence and avoid representing the time element in a computation. The real-time schema the authors introduce provides both these features.

Definition 1. A real-time computation schema is a triple S=(M,Ω,T) where:

i) M is a finite set of named memory locations $\{m_0,m_1...m_n\}$,
ii) Ω is a finite set of elementary operations (mappings) $\{a_1,a_2...a_k\}$.

Associated with each $a_i \varepsilon \Omega$ are:

a) $Da_i \subseteq M$, the domain locations;
b) $Ra_i \subseteq M$, the range locations;
c) $\gamma_i \varepsilon m$, a condition flip-flop used to determine whether an operations jump successor $a_i(\gamma_i=0)$, or the fall through successor $a_i(\gamma_i=1)$ should be enabled.

iii) T is the control graph and $T = \{Q, q_0, Q_T, \Sigma, \mu, \delta\}$ where:
- a) Q is a finite set of states $\{q_0, q_1 \ldots q_k\}$
- b) $q_0 \epsilon Q$ is the unique initial state;
- c) q_I is the interrupt state
- d) $Q_T \subseteq Q$ is a set of acceptor states;
- e) Σ is an alphabet corresponding to the allowed set of operations, $\Sigma = \{\sigma_i \mid a_i \epsilon \Omega\}$
- f) μ is a LIFO stack
- g) δ is a partial transition function $\delta: Q \times \mu \times \Sigma \rightarrow Q \times \mu$

The sequence of allowed operations, corresponding to a computation in realtime, is enforced by the control graph T. The nodes in the graph represent the state of the system's memory and transitions from one state to the next in discrete time by the directed edges between nodes. Construction of a real-time schema control graph for a given process is based on a program graph representing the algorithmic part of the computation. The graph must allow interrupt capability at each node.

The control graph for an interrupt driven process would not be finite state without use of an interrupt queue.

Construction of a real-time schema control graph for a given process is based on a program graph, similar to a flowchart, that represents the algorithmic part of the computation. The process allows interrupts at any node which are themselves processes. Multiple interrupts will be given priority by some preset queue discipline. Interrupt jumps and returns are handled in the interrupt state q_I.

Figure 1 illustrates the interrupt state for a system with two interrupt processes. Since in general, the actual interrupt jump and return are handled by hardware, the transition to and from the interrupt state from any interrupted state q_i other than q_I is shown by a dotted bi-directional arc and has no labeled software operation. Once in the interrupt state the appropriate interrupt process is determined by polling, or any other means of identifying the interrupt and if the interrupt process can be enabled, the interrupted state is pushed onto a return state stack μ and the interrupt process is then initiated. When complete, a transition is made to the interrupt state where the return state is popped from the stack μ and control resumes at that state.

Example 1. Consider a simple real-time process corresponding to an interrupt driven, one-step table lookup procedure. The system has one input (i_b) and one output channel (o_b), and a small memory $M = \{m_0, m_1 \ldots m_{10}\}$. Once a request is initiated by an interrupting process on a specified I/O channel, all further requests to that channel will be disabled until the original request is satisfied. Note that even for this simple one-step example a program graph could become quite complex when nested levels of interrupts are considered. The graph allows the possibility of a countably infinite number of interrupts to

occur. A situation where this is possible is when a computation becomes I/O bound. Hence the real-time program graph representing the total process behavior is not finite state. To simplify the graph the process can be modularized in a manner that divides the graph into two identifiable sub-graphs, an algorithmic process and the remaining interrupt processes. Figure 1 illustrates this simplification for the table lookup example. This structure is equivalent to the more general structure described above if each node generated is augmented by the interrupt subgraph.

More generally, any real-time process can be represented in a modular fashion, by identifying independent sub-processes in both the algorithmic and interrupt process subgraphs. More generally, any real-time process can be represented by a main control subgraph and one or more interrupt subgraphs.

Interpretation of a Schema. In order to assign "meaning" to the evaluation of a calculation within a particular schema, a precise interpretation must be specified. The idea of an interpreted schema is similar to that of Karp and Miller [K1] with the addition of the notion of time.

Definition 2. An interpretation I of schema S with respect to time is specified by:

i) An initial time t_0, and discrete ordered, but not necessarily uniform time steps $t_1,\ldots t_k,\ldots$

ii) A function C which associates with each element $m_i \epsilon M$, a set $C(m_i)$, the possible contents of m_i. The notation $[m_i(t_j)]$ represents a specific content of location m_i at time t_j, $[m_i(t_j)] \epsilon\ C(m_i)$ and $[M(t_j)]$ the total contents of memory at time t_j, $[M(t_j)] \epsilon \bigtimes_{m_i \epsilon M} C(m_i)$

iii) An initial contents of memory $[M(t_0)]$

iv) For each $a_i \epsilon \Omega$

a) the time to execute operation $a_i, \tau(a_i)$

b) a function

$$F_{a_i} : \bigtimes_{m_j \epsilon D_{a_i}} C(m_j) \rightarrow \bigtimes_{m_j \epsilon R_{a_i}} C(m_j)$$

v) A function

$$G: \bigtimes_{m_i \epsilon M} C(m_i) \rightarrow \bigtimes_{\gamma_i \epsilon M} C(\gamma_i)$$

The function F_{a_i} determines the functional outcome of an operation a_i, i.e. it stores the result in the range specified. Upon completion of operation a_i, the function G performs a test on the associated condition flipflop γ_i to determine whether the jump successor or fall through successor operation is enabled next.

Table 1 gives an interpretation of operations for Example 1. The condition flip-flops are set as follows: $\gamma_1=1$ iff $[m_0]=\gamma$ otherwise $\gamma_2=\phi$ and $\gamma_2=\gamma_3=\gamma_4=\gamma_5=\gamma_6=\gamma_7=\gamma_8=1$. The set of acceptor states $\{Q_T\}=\{q_1\}$. The interrupt priority is defined as last request, however an interrupt may not interrupt itself. For example, the operation specified by $\delta(q_j,\mu,a_3)$ is not enabled whenever the top element of the return state queue μ_1 is q_2 or q_3 and similarly $\delta(q_i,\mu,a_6)$ is not enabled if μ_1 is q_4 or q_5.

This is a simple example, however, it is representative of any multi-step algorithm which calculates a specific output or outputs from a single or block input without producing partial results.

A Time Description of Process Behavior. In order to develop a description of the time behavior of a process, it is useful to classify the operation set.

Definition 3. The operation set Ω is partitioned in six distinct classes $\Omega=(A,B,\Gamma,\Delta,E,Z)$

Where

i) $A=\{a_i\varepsilon\Omega|D_{a_i}=i_b,\gamma_i=1\}$ is the set of input operations,

ii) $B=\{a_i\varepsilon\Omega|o_b=R_{a_i},\gamma_i=1\}$ is the set of output operations,

iii) $\Gamma=\{a_i\varepsilon\Omega|\gamma_i E\{0,1\}\}$ is the set of conditional jump or branch operations. If $\gamma_i=1$, the fall through condition is met and operation a_i is enabled otherwise $\gamma_i=0$, the branch condition is met and operation a_i' is enabled.

iv) $\Delta=\{a_i\varepsilon\Omega|i_b\not\varepsilon D_{a_i},o_b\not\varepsilon R_{a_i},\gamma_i=1\}$ is the set of data transformation operations, including arithmetic, logic, move and modify operations.

v) $E=\{a_i\varepsilon\Omega|\delta(q_i,\mu,a_i)=(q_k\cdot q_j\downarrow\mu),\gamma_i=1\}$ is the set of interrupt and subroutine jump operations*,

vi) $Z=\{a_i\varepsilon\Omega|\gamma_i\varepsilon\{0,1\}\}$ is the set of conditional interrupt return operations if $\gamma_i=1$, the partial transition function $\delta(q_j,\mu,a_i)=(q_k,\uparrow\mu)$ determines the next enabled operation via the fall through operation a_i, otherwise $\gamma_i=0$ and the branch operation a_i causes a system reset to state q_0,

Processes in a modular structure may depend on other processes or external input for their initiation. This may take the form of a flag system that will not allow a process to continue until certain input conditions are satisfied. To symbolize this condition on the schema graph we use a double arrow leading into the node where such a condition is set (Figure 2a). The criteria we have set which will determine when a process yields useful information is the reaching of an acceptor state of the process. This event usually signals the availability of an output and will be symbolized on the schema graph with a double arrow leading out of a node (Figure 2b).

An Instantaneous Description. Once the distinction between operation classes has been introduced, some way of representing the instantaneous system behavior with respect to a

specific string of operations must be developed. The concept of an instantaneous description used by Karp and Miller [K1] seems appropriate.

Definition 4. An initial instantaneous description is a quadruple $\theta(t_0) = (q_0,\psi(t_0)),[M(t_0)],\mu]$ where q_0 is the initial state, $\psi(t_0) = \{a_i \varepsilon \Omega | \delta(q_0,\mu,a_i)$ is defined and $\gamma_i=1\}$ is the initially enabled operations. $[M(t_0)]$ is the initial memory contents and $\mu \neq 0$ is the initial LIFO stack state.

For a specific sequence $x \varepsilon \Sigma^*$, $x=a_{i1}a_{i2}\ldots a_{ik}$, the time to execute any subsequence $a_{i1},a_{i2}\ldots a_{ij}$ is given by:

$$t_j = t_0 + \sum_{\ell=1}^{j} \tau(a_{i\ell})$$

An instantaneous description at time t_j during the execution of sequence x is given by a quadruple $\theta(t_j)=\{q_\ell,\psi(t_j),[M(t_j)],\mu\}$ where:

i) q_ℓ is the present state
ii) $\psi(t_j)$ is the set of enabled operations
iii) $[M(t_j)]$ is the memory contents
iv) μ is the LIFO stack state.

The instantaneous description $\theta(t_j)$ can be defined recursively.

Definition 5. Given a sequence $x \varepsilon \Sigma^*$, $x=a_{i1}a_{i2}..a_{ik}$ and the instantaneous description after execution of a $a_{ij}, j \leq k$, $\theta(t_j) = \{q_\ell,\psi(t_j),[M(t_j),\mu\}$, a consecutive description $\theta(t_{j+1})=\theta(t_j+\tau(a_{ij+1}))=\{q'_\ell,\psi(t_{j+1}),[m(t_{j+1})],\mu'\}$ is defined iff $\delta(q_\ell,\mu,a_{ij+1})$ is defined. If so,

i) $(q'_\ell,\mu') = \delta(q_\ell,\mu,q_{ij+1})$

ii) $[M(t_{j+1})] = F_{a_j}[M(t_j)]$

iii) $\psi(t_{j+1}) = \{a_i \varepsilon \Omega | \delta(q'_\ell,\mu',a_i)$ is defined where
$[(\gamma_1,\gamma_2,\ldots,\gamma_n)] = G([M(t_{j+1})])$

Properties of the Schemata. A real-time computation corresponds to a well defined sequence of operations, that begins in the initial state and either iterates through or terminates in some designated acceptor state. An acceptor state in the graph model, represents a logical point in the computation where the calculation on input information is in an acceptable condition for output. This may correspond to a terminal state in a fixed length algorithm or an appropriate sample state in an iterative algorithm. Acceptability is a function of the "useful information" calculated in a process.

*The notation $q_i \downarrow \mu, \uparrow \mu, \mu = q_k \cdot \mu'$ represent "pushing" q_i on stack μ, "popping" stack μ and q_k is the top element of μ respectively.

Definition 6. A string $\sigma\varepsilon\Sigma^*$ is a well defined sequence of operations (w.d.s.) for a schema S is all prefixes β of σ, $\delta(q_0,\mu,\beta)$ is defined on S, where $\mu(t_0)=\emptyset$.

If $\ell(\sigma)$ of the string σ, e.g. if $\sigma=a_{i1},a_{i3}\ldots a_{ik}$, $\ell(\sigma)=k$, then the following may be stated.

Definition 7. The set of k-length well defined sequences for a schema S is a finite set $\Sigma^K=\{\sigma\varepsilon\Sigma^* | \ell(\sigma)=k \text{ and, } \sigma \text{ is well defined}\}$.

Definition 8. Let $\sigma_i=a_{i1}\ldots a_{ik}$ be a k-length sequence in Σ^k for a schema S. The ordered set of states $q_i = q_0,q_{i1}\ldots q_{ik}$, of length k+1 is the q-set associated with the computation σ_i, where q_0 is the initial state and for $i\leq j\leq k$, $q_{ij}\varepsilon Q$, $q_{ij}\varepsilon\sigma_i$ and $\delta(q_{ij-1},\mu,a_i)=(q_{ij},\mu')$

Definition 9. A useful calculation occurs in a sequence $\sigma_i\varepsilon\Sigma^K$ under schema S if $\{q_i\}$, the associated q-set, contains at least one element $q_i\varepsilon Q_T$, the set of acceptor states.

The q-set concept is essential for the analysis of useful information production by a computation. For any sequence of operations, an associate q-set may be generated. An a-posteriori analysis of this q-set with refer ence to the occurences of acceptor states reflects the useful information

In the example of Figure 1, $\sigma_0=a_1a_3a_4a_5a_1'a_6a_7a_8$ is a well defined sequence. The associated q-set is $\{q_0\}=\{q_0q_0q_2q_3q_0q_1q_0q_4q_5q_0\}$ and a useful calculation is performed when state q_1 is reached.

Earlier definitions introduce the concept of length bounded strings. In order to develop a notion of time bounded strings and information production rates, the following are defined.

Definition 10. The set of well defined sequences bounded by time t is defined $\Sigma_t=\{\sigma\varepsilon\bigcup_i\Sigma^i | \tau(\sigma)\leq t\}$

It is well known that both Σ^k and Σ_t form a language of regular expressions.

Definition 11. A trace on Σ_t is a set of q-sets $T_t=\{\{q_i\}|\sigma_i\varepsilon\Sigma_t\}$.

Definition 12. An incremental trace over time interval $T_{t,\hat{\tau}}=\{\{\nu\}|\{q_i\}\varepsilon T_{t-\hat{\tau}},\{q_i,\nu\}\varepsilon T_t\}$ where $\{q_i,\nu\}$ is a q-set which consists of the ordered set $\{q_i\}$ followed by the ordered set of states $\{\nu\}$.

Examination of the elements of $T_{t,\hat{\tau}}$ will indicate generation of useful information during the time interval $(t-\tau,t)$. As was suggested above, useful information depends on the occurrence of acceptor state in a particular $\{\mu\}$. Mere occurrence of an acceptor state does not in itself guarantee the generation of useful information, but serves to indicate that information was calculated.

Information Flow In A Schema. The q-set and incremental trace concept provide a means for determining when a calculation has occurred for a given w.d.s. σ. Since the definition of real-time implies non-reproducible inputs, some way of gauging whether input data is utilized completely or lost is necessary.

Given a time t, a schema S and a set of time bounded w.d.s. Σ_t, a set of strings $\hat{\Sigma}_t \subset \Sigma_t$ is said to be input-preserving if the information input, as represented by a particular string, is available for all operations for which it is intended. The "definition" is purposely vague, for it depends completely on the algorithm and the environment which input information is important. Obviously, a string which includes two successive input operations with the same range locations is not input preserving unless the input information is not needed for any operation. The presence or absence of a "useful" calculation, as defined by the occurrence of acceptor states within the q-set for a string or incremental trace for the time interval between two inputs in a string, does not guarantee that input is or is not preserved. However, by examining the input-output relationship between operations some sufficient conditions for strings to be input preserving can be derived.

Definition 13. A kxk dependency matrix at time t for a schema S with operation set $\Omega=\{a_1,a_2,\ldots,a_k\}$ is $\Lambda^t[\lambda^t_{ij}]_{\substack{i=1,\ldots,k\\ j=1,\ldots,k}}$ where $k_{ij}=1$ iff

i) $R_{a_i} \cap D_{a_j} \neq \emptyset$

ii) There exists a $\sigma \in \Sigma^t$ such that for $x,y,z \in \Sigma^*$
   1. $\sigma = x a_i y a_j z$ and,
   2. For all $y' \in \Sigma^*$, $a_m \in \Omega$ such that $y' a_m$ is a prefix of y, $R_{a_m} \cap R_{a_i} \cap D_{a_j} = \emptyset$

Hence, an operation a_j is said to depend on a_i if a_i possibly produces information used by a_j and for some w.d.s. σ, a_j does not use that information. From well known graph theoretic techniques the set of operations dependent on a particular operation a_i over time t, $D^t_{a_i}$ can be determined since Λ^t is a restricted incidence matrix for the graph (Ω,H^t) where $H^t(a_i)=a_j$ iff $\lambda^t_i = 1$. $D^t_{a_i}=\{a_j \in \Omega |$ there is a path from a_i to a_j in $(\Omega,H^t)\}$.

An operation $a_i \notin$ ΓUEUZ is trivial (processes no information) if $D^t_{a_i}=\emptyset$.

Remark: Sufficient conditions for $\sigma \in \Sigma^t$ to be input preserving are: For all $x \in \Sigma^*$, $a_i \in A$, such that xa_i is a prefix of σ, a_i nontrivial

i) There exists a $y \in \Sigma^*$, $a_j \in D^t_{a_i}$ such that xa_iya_j is a prefix of and $\Sigma(q_0,\emptyset,xa_iya_i) \in Q$, and

ii) For all $y' \in \Sigma^*$ such that either $\sigma = xa_iy'$ or $xa_iy'a_k$, $a_k \in A$ is a prefix of σ and for which there is no y'' for which $y''a_\ell$ is a prefix of y', $a_\ell \in A$, $R_{a_i} C R_{a_\ell}$, there exists no $z \in \Sigma^*$, $a_m \in D^t_{a_i}$ for which $\delta(q_0,\emptyset,xa_iy'za_m) \in Q_T$.

The condition requires that at least one useful calculation on each nontrivial input be performed and furthermore that all defined calculations are performed by the end of the sequence or prior to any other input which writes on the same range memory locations.

The other vital concern in a real-time system is that output information is available, when needed, to implement positive control. Information flow is said to occur for some $\sigma \in \Sigma^t$ $\sigma = a_{i1}a_{i2}..a_{ik}$ under schema S, if the information input by some nontrivial operation ($a_{ij} \in A$) is processed, either partially or completely ($a_{i\ell} \in D^t_{a_{ij}}$, $\ell > j$, $\delta(q_0,\emptyset,a_{i1}\ldots a_{i\ell}) \in Q_T$) and is output ($a_{\ell m} \in B, m>\ell$, $R_{a_i} \cap D_{a_{im}} \neq \emptyset$)

An exact definition of information flow is difficult to state in general, since much depends on the input and output interrupt rates, the algorithm and other factors. However, occurrence of information flow can be related back to the incremental trace concept.

Remark: A sequence $\sigma \in \Sigma^t$, $\sigma = a_{i1},a_{i2}\ldots a_k$, represents information flow only if some time t_j, $a_{ij} \in B$ and for some time t_ℓ, $a_{i\ell} \in A$, $t_\ell < t_j$, $\in T_{t_j,t_j-\ell t}$, where ν is incremental trace on σ, ν $Q_T \neq \emptyset$.

Input preserving sequences, as loosely defined above, are sequences for which input information is completely processed as defined by the algorithm represented by the schema. Sequences which represent maximum information flow would necessarily be input preserving. The output symbols within these sequences must occur at appropriate intervals such that the processed information is output.

An Example of a Complex Process. In many real-time applications complex algorithms are realized by the computer subsystems which may consist of many component algorithms. The previous example given had only one component, yet the schemata as defined has the capacity to model processes of much greater complexity.

The basic schema graph can be partitioned into several subgraphs $P_{\emptyset}$, $P_1,...P_m$, P_{I1}, $P_{I2},...P_{In}$ where P_o, $P_1...P_m$ are structured process module partitions of the main process algorithm and $P_{I\emptyset}$, $P_{I1}...P_{In}$ are similar constructs based on the interrupt processes. The ordering of partitions in the interrupt sub-structure can be defined according to the interrupt priorities assigned to a specific interrupt module.

An alternate description of a structured modular process could be based on process flow, in terms of sequential ordering of processes according to some control structure and priority of processes.

Definition: A structured process P is a quadruple (q,P,QT,π) where:

i) q is the initial information state of the process, this can be the contents of memory.
ii) p is a regular expression of operations that describes all sequences of operations allowed in the process.
iii) Q_T is the state or set of states in which the process terminate.
iv) π is the process priority.

We specify that a simple structured process begins in an initial state q_o and each subsequent state, in time, is a function of an enabled operation at that time and the present state of the system. If the process terminates, there exists a final state where no further operations are enabled, otherwise there are no terminal states.

A structured complex process is defined in an analogous manner. Decomposition of a complex task into structured modules was discussed in an earlier section of this paper. To allow dependent modules we structure the independent parts such that the final state of one process is the initial state of another. An interrupt process, which has a dynamic rather than fixed initial state, can be viewed in a similar fashion, i.e. an interrupt process begins in the state from which it was interrupted.

Example 2. Figure 5 is a flow chart description of a typical radar tracking algorithm that might be part of some real-time computer application.

A structured modular decomposition of the process in a Top-Down manner might yield a process flow as in Figure 6. Each sub-process yields a schema diagram, Figure 4 with an interpretation of operations in Table 2. A regular language descriptor of the process of Figure 6 is: $P=[P_0P_1P_2P_3(P_1P_2P_3)^*P_4((P_1P_2P_3)^*P_4)^*P_4(((P_1P_2P_3)^*P_4)^*P_5)^*P_6P_1P_2P_3P_4P_5(P_1P_2P_3P_4P_5)^*]^*$ Each structured process module is described by the previously defined convention $P = (q,p,Q_T,\pi)$ in Table 3.

An interpretation of operations in given for this example in Table 2. In contrast to the previous example, where a specific range and domain was specified for each operation, there are no entries. At this level of conceptualization, machine dependency has not been encountered. The process can be interpreted in such a manner if the Top-Down decomposition had continued to deeper levels. At this level, however, the process structure is apparent and input-ouput relations are known.

Extensions and Conclusions. In the above example, a single processor was assumed to implement the algorithm. The interrupt handler was assumed to be a hardware function of the processor. This is not a rigid interpretation. With the advent of inexpensive microprocessors, one can conceive of a hardware implementation which distributes the processes over several microprocessors. Two configurations come to mind immediately, a system where each process is assigned to a different processor and a system where the main process is implemented on a single processor, the interrupt handler is implemented as a message handling communication concentrator and each interrupt process is associated with a microprocessor based data acquisition or control device.

In the most general case, the use of a separate processor for each main processor is perhaps wasteful of processor hardware unless an amount of parallelism in execution is assumed. Since the schemata can handle the analysis of parallel processes it provides a tool for comparing the capacity of single and multiprocessor implementations of real-time processes.

For the second case, the interrupt handler becomes a hardware implementation of an interprocess message facility. Studies can be made of process driven real-time systems with this interpretation.

A useful extention would be to consider the variety of possible hardware implementations of a process schemata. This would allow an optimal design to be developed or provide a set of alternative architectures to provide a level of graceful degradation and fault tolerance.

Bibliography

A1 W. R. Adrion, P. A. Frick and P. A. Szulewski, “Analysis of Real Time Systems: An Information Theoretic Approach,” Proceedings of the 13th Annual Allerton Conference on Circuit and System Theory, October, 1975.

A2 _____, “Real Time Schemata,” Proceedings of the Ninth Annual Asilomar Conference on Circuits, Systems and Computers, November, 1975.

D1 O. J. Dahl, E. W. Dijkstra and C. A. R. Hoare, Structured Programming, Academic Press, 1972.

K1 R. M. Karp and R. E. Miller, “Properties of a Model for Parallel Computation: Determinancy, Termination, Queueing,” SIAM Journal of App. Math., (Nov. 1966) 14, 1390-1411.

Y1 Y. I. Yanov (Ianov), “The Logical Scheme of Algorithms,” Problems in Cybernetics, Vol. 1, Pergamon Press, 1960, 82-140.

OP	Class	D_{a_i}	R_{a_i}	Condition, Action γ=1	γ=0
a_1	Γ	$m_0, m_2 \ldots m_9$	m_1, m_0	$[m_0]=0$, ___	$[m_0]\neq 0, m[m_0]\rightarrow m_1$ $0\rightarrow m_0$
a_2	Γ			unc, ___	___
a_3	E		μ	___, $\downarrow\mu$	___
a_4	A	i_b	m_0	___, $[i_b]\rightarrow m_0$	___
a_5	Z	μ		___, $\uparrow\mu$	___
a_6	E		μ	___, $\downarrow\mu$	___
a_7	B	m_1	o_b	___, $m_1\rightarrow o_b$	___
a_8	Z	μ		___, $\uparrow\mu$	___

Table 1. Interpretation of Operations (Ex. 1)

OP	Class	D_{a_i}	R_{a_i}	Action γ=1	γ=0
a_0	Δ			init. memory	
a_1	Γ			scan data avail.	not avail.
a_2	Γ			scan complete	not complete
a_3	Γ			target in range	not
a_4	Γ			enemy	friend
a_5	Γ			enemy in range	not
a_6	Γ			missile fired	not
a_8	Γ			update traj.	no
a_9	Δ			update	___
a_{10}	Δ			launch	___
a_{11}	E			interrupt, $\downarrow\mu$	___
a_{12}	A			input scan data	___
a_{13}	Z			return, $\uparrow\mu$	reset return
a_{14}	B			output m. control	___
a_{15}	A			input m. status	___

Table 2. Interpretation of Operations (Ex. 2)

Process	Init. State	Regular Expression Description	Terminal St.	Priority
P_0	q_0	(a_0)	q_1	π_0
P_1	q_1	$(\acute{a_1}^* a_1)$	q_2	π_1
P_2	q_2	$(\acute{a_2}^* a_2)$	q_3	π_2
P_3	q_3	$(a_3+\acute{a_3})$	q_1, q_4	π_3
P_4	q_4	$a_4+\acute{a_4}$	q_1, q_5	π_4
P_5	q_5	$(\acute{a_5}+a_5(\acute{a_6}+a_6(\acute{a_8}+a_8a_9))$	q_1, q_{10}	π_5
P_6	q_{10}	(a_{10})	q_1	π_6
P_{I1}	q_I	$(a_{11}a_{12}a_{13})$	q_I	π_7
P_{I2}	q_I	$(a_{11}a_{14}a_{13})$	q_I	π_8
P_{I3}	q_I	$a_{11}a_{15}(a_{13}+\acute{a_{13}})$	q_I	π_9

Table 3. Process Description (Ex. 2)

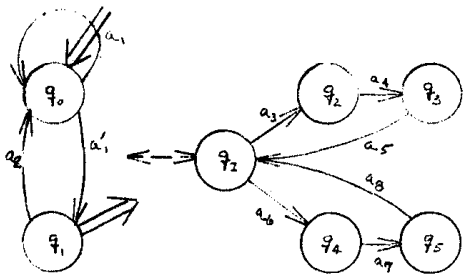
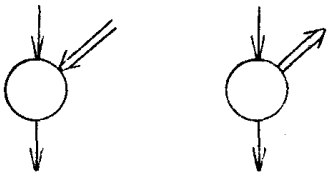
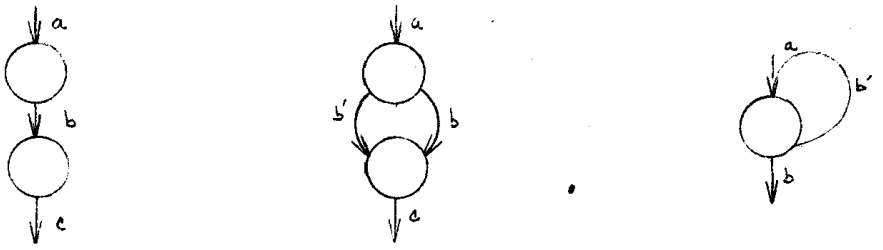


Figure 1



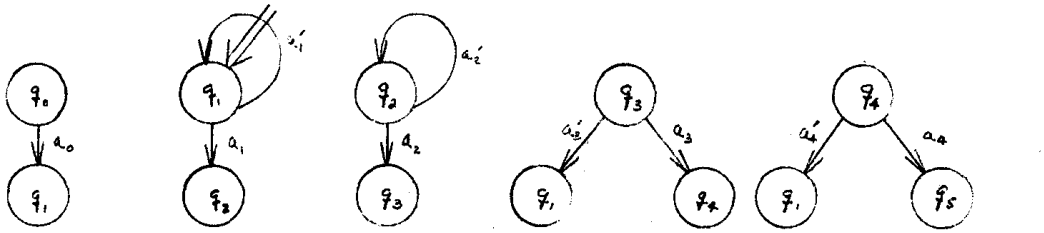
(2a) (2b)

Figure 2

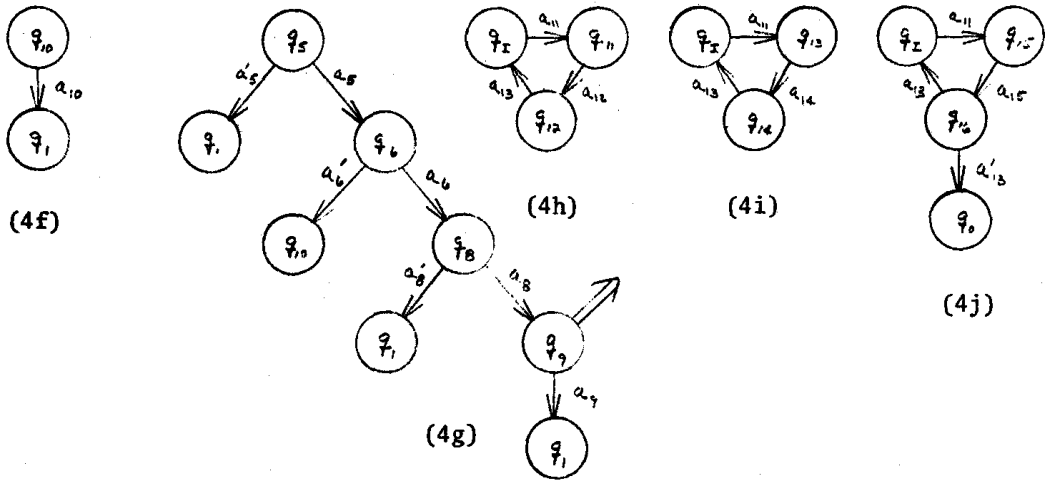


(3a) (3b) (3c)

Figure 3



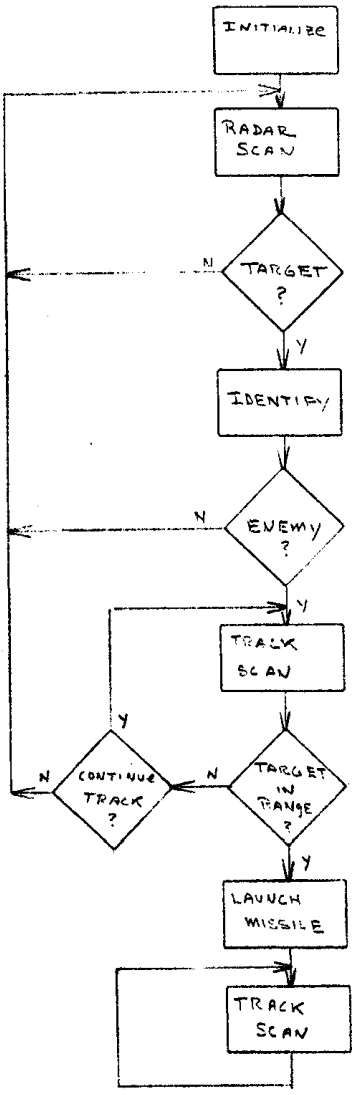
(4a) (4b) (4c) (4d) (4e)



(4f) (4h) (4i) (4j)

(4g)

Figure 4



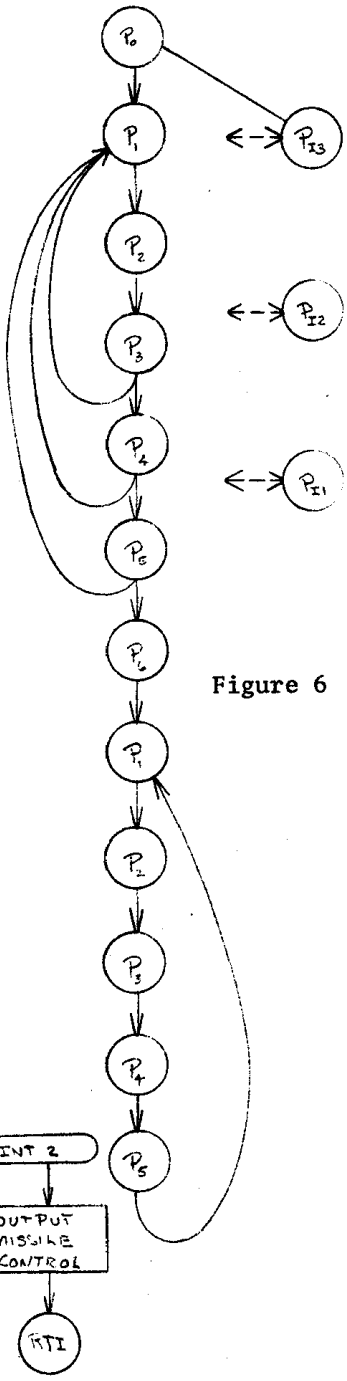
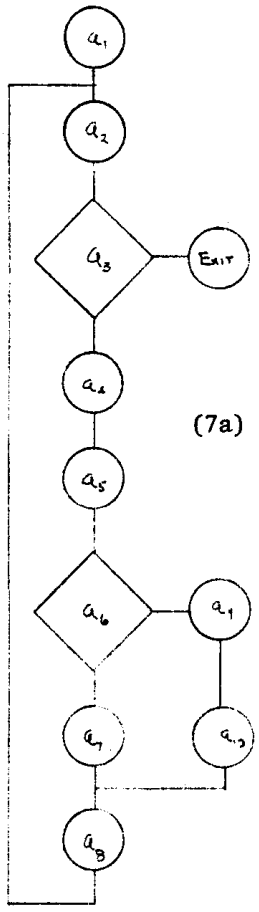
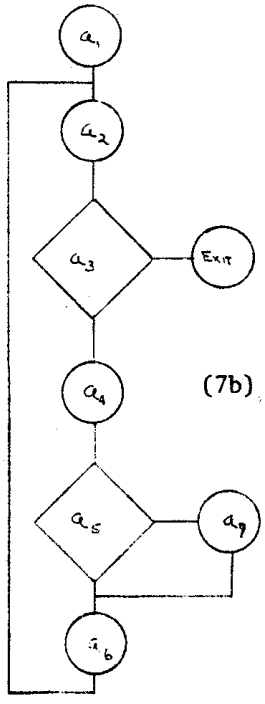


Figure 6



(7a)

Figure 7



(7b)

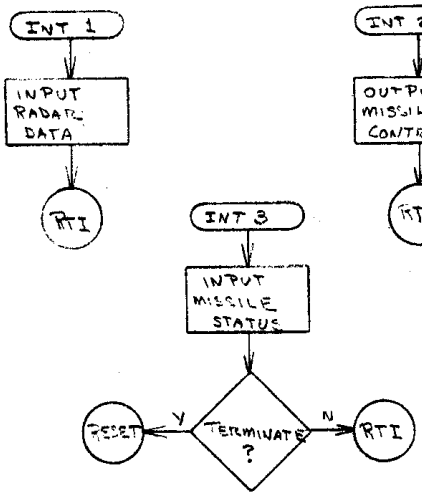


Figure 5