N89-26584

Parallel Plan Execution with Self-Processing Networks

C. Lynne D'Autrechy and James A. Reggia*

Department of Computer Science
University of Maryland, College Park, MD 20742

*Also with the University of Maryland Institute for Advanced Computer Studies
and the Department of Neurology, University of Maryland

Abstract: A critical issue for space operations is how to develop and apply advanced automation techniques to reduce the cost and complexity of working in space. In this context, it is important to examine how recent advances in self-processing networks can be applied for planning and scheduling tasks. For this reason, we are currently exploring the feasibility of applying self-processing network models to a variety of planning and control problems relevant to spacecraft activities. Our goals are both to demonstrate that self-processing methods are applicable to these problems, and that MIRRORS/II, a general purpose software environment for implementing self-processing models, is sufficiently robust to support development of a wide range of application prototypes. Using MIRRORS/II and marker passing modelling techniques, we implemented a model of the execution of a "Spaceworld" plan which is a simplified model of the Voyager spacecraft which photographed Jupiter, Saturn, and their satellites. This study demonstrates that plan execution, a task usually solved using traditional AI techniques, can be accomplished using a self-processing network. The fact that self-processing networks have been applied to other space-related tasks in addition to the one discussed here demonstrates the general aplicability of this approach to planning and control problems relevant to spacecraft activities. This work also demonstrates that MIRRORS/II is a powerful environment for the development/evaluation of self-processing systems.

I. Introduction

A critical issue for space operations is how to develop and apply advanced automation techniques to reduce the cost and complexity of working in space. In this context, it is important to examine how recent advances in self-processing networks (connectionist models, artificial neural networks, marker passing systems, etc. [3]) can be applied to planning and scheduling tasks. Most successful work with such models has focused on fairly low-level applications (pattern recognition or completion, associative memory, constraint satisfaction, etc.) and relatively little has been done in traditional AI problem-solving areas like planning. Thus, while these methods potentially offer tremendous advantages for complex automation applications (massively parallel processing, fault tolerance, etc.), it is currently difficult to see how they can be adopted directly.

For this reason, we are exploring the feasibility of applying self-processing network models to a variety of planning and control problems relevant to spacecraft activities. Our goals are both to demonstrate that self-processing methods are applicable to these problems, and that MIRRORS/II, a general purpose software environment for implementing self-processing models [1,2], is sufficiently robust to support development of a wide range of application prototypes. While a number of specific applications have recently been developed using MIRRORS/II for spacecraft applications (camera controller [5], diagnostic problem-solver [6], etc.), this paper focuses on a specific plan execution example.

II. Self-Processing Network Models and Marker Passing

To enable the reader less familiar with current work on self-processing network models to follow the principal ideas embodied in MIRRORS/II, we introduce some basic concepts and terminology, simplifying somewhat for brevity. The term *self-processing network model* as used in this paper refers to models in which many, usually simple, processing elements operate in parallel and communicate via connections (links) between nodes. For our purposes, it is convenient to view self-processing network models as having two components: a network and an activation method. The *network* consists of a set of processing *nodes* connected together via *links*. Nodes directly connected to one another are said to be *neighbors* of each other. The *activation method* is a *local* rule or procedure that each node follows in updating its current state in the context of information from neighboring nodes. Typically, the goal in constructing and running a simulation with a self-processing network model is to demonstrate that some *global* behavior (behavior of the network as a whole) can emerge from the concurrent *local* interactions between neighboring nodes during a simulation.

A great number of self-processing network models have been proposed and studied since the 1940's in cognitive science, artificial intelligence, and neurophysiological modelling [3]. This paper is only concerned with one class of such models referred to as *marker passing systems*. The networks in these symbol-processing models usually have semantically-labeled, unweighted links and implement spreading activation by passing symbolic (non-numeric) labels or markers. Typically, a node might have a dozen marker bits ($M1$, $M2$, ..., $M12$), binary switches that can be turned on or off when the appropriate marker is "received." Using these markers, such networks provide powerful mechanisms for implementing set operations (intersection, union, etc.) as well as some forms of deduction (transitive closure of relations, inheritance of properties, etc.).

III. The MIRRORS/II Simulator

MIRRORS/II is an extensible general-purpose simulator which can be used to implement a broad spectrum of self-processing network models. MIRRORS/II is distinguished by its support of a high-level non-procedural language, an indexed library of networks, spreading activation methods, learning methods, event parsers and handlers, and a generalized event-handling mechanism.

The MIRRORS/II language allows relatively inexperienced computer users to express the structure of a network that they would like to study and the parameters which will control their particular self-processing network model simulation. Users can select an existing spreading activation/learning method and other system components from the library to complete their model; no programming is required. On the other hand, more advanced users with programming skills who are interested in research involving new methods for spreading activation or learning can still derive major benefits from using MIRRORS/II. The advanced user need only write functions for the desired procedural components (e.g., spreading activation method, control strategy, etc.). A specification file, written in the MIRRORS/II language by the user, serves as input to MIRRORS/II.

Self-processing network models developed using MIRRORS/II are not limited to a particular processing paradigm. Many spreading activation methods and learning methods including Hebbian learning, competitive learning, and error back-propogation are among the resources found in the MIRRORS/II library. MIRRORS/II provides both synchronous and asynchronous control strategies that determine which nodes should have their activation values updated during an iteration. Users can also provide their own control strategies and have control over a simulation through the generalized event handling mechanism.

Simulations produced by MIRRORS/II have an event-handling mechanism which provides a general framework for scheduling certain actions to occur during a simulation. MIRRORS/II supports system-defined events (constant/cyclic input, constant/cyclic output, clamp, learn, display and show) and user-defined events. An event command (e.g., the input-command) indicates which event is to occur, when it is to occur, and which part of the network it is to affect. At run time, the appropriate event handler performs the desired action for the currently-occurring event.

MIRRORS/II was originally designed for implementing connectionist models with no significant thought being given to how marker-passing methods might be developed in the context of MIRRORS/II. Thus, at the start of the work described here, it was not immediately obvious whether MIRRORS/II could support marker passing methods without alterations.

IV. Non-Hierarchical Plan Execution

Using MIRRORS/II and marker passing techniques, we implemented a model of the execution of a "Spaceworld" plan as described in [4]. Spaceworld is a simplified model of the Voyager spacecraft which photographed Jupiter, Saturn, and their satellites. The specific Spaceworld plan used here describes the sequential and parallel sequence of steps (goals) which must be taken by the Voyager spacecraft in order to photograph two satellites and transmit the photographs to earth. Each goal in the plan has the following

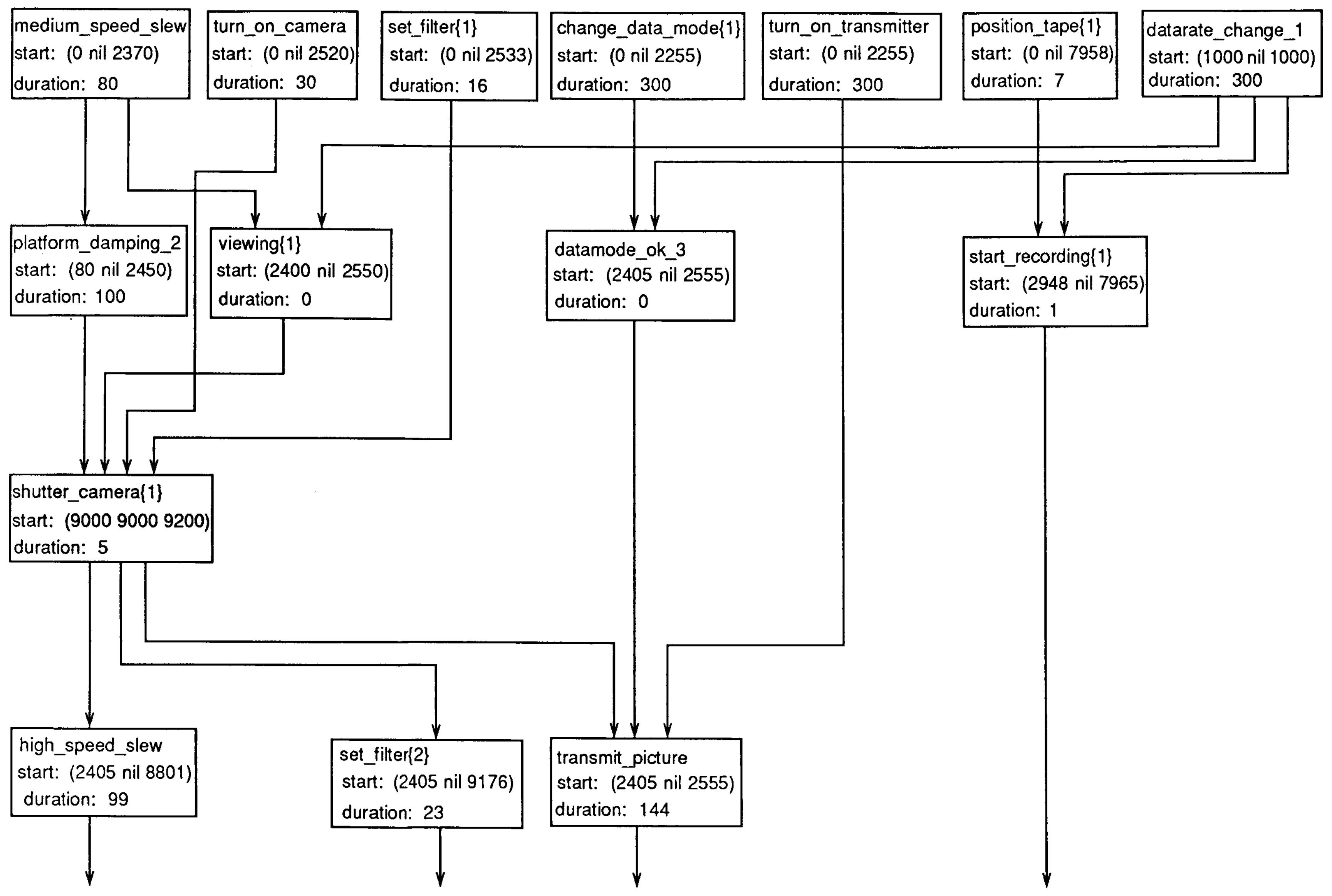

Figure 1: The beginning portion of Vere's non-hierarchical Spaceworld plan.

four parameters: an earliest starting time, an ideal starting time, a latest starting time, and a duration. Often, the ideal starting time parameter does not exist; in this case, the earliest starting time is also the ideal starting time. A portion of this specific Spaceworld plan is illustrated in Figure 1. Each box in the figure represents a goal in the plan. The first line in a box is the name of the goal. The second line labelled "start:" indicates the early, ideal, and latest starting times in seconds respectively. The third line labelled "duration:" indicates the duration in seconds. The arrows in the plan represent dependency relationships or the flow of execution. For example, the platform_damping_2 goal must finish executing before the shutter.camera{1} goal can begin executing.

A goal in the plan cannot be executed until two constraints have been satisfied - the dependency constraint and the starting time constraint. First, all goals which precede a given goal G in time must have finished executing before G can begin executing. Once this dependency constraint has been satisfied, the starting time constraint must be satisfied. Optimally, a goal should begin execution at its ideal starting time or, if that is not possible, at the earliest time thereafter up to and including the latest starting time. If the latest starting time is reached and the dependency constraint has not yet been satisfied then that goal will never be executed.

The MIRRORS/II specification of the self-processing network used to implement the execution of the Spaceworld plan discussed in [4] is pictured in Figure 2. Each goal in the plan is represented as a node in a self-processing network (all statements beginning with "[node" and ending with "]"). All the goals in the plan share the same parameters so their corresponding nodes are grouped into a single set (lines 1-22). Each node has two user-defined attributes - *start* and *duration*. The start attribute is a triple of values representing the earliest, ideal, and latest starting times for a goal in the plan. For example, the triple representing the earliest, ideal, and latest stopping times respectively for node shutter_camera{1} is (2400 2500 2550). All times in the plan are given in seconds relative to the starting time of the plan. The goal duration is a number representing the time in seconds it takes to complete the execution of the goal. Connections between the nodes represent the time sequence dependency relationships. For example, the statement [node turn_on_transmitter ... (plan transmit_picture)] in Figure 2 indicates that node turn_on_transmitter connects to node transmit_picture. This connection indicates that execution of the turn_on_transmitter goal must be completed before the execution of the transmit_picture goal can begin.

```
;--- Set containing all the goals (nodes) in a non-hierarchical Spaceworld plan

[set plan (contains
        calibrate_gyros change_data_mode{1} change_data_mode{2}
        change_data_mode{3} change_data_mode{4} clear_tape
        consolidate_tape{1} consolidate_tape{2} datamode_ok_1
        datamode_ok_2 datamode_ok_3
        datamode_ok_4 datamode_ok_5 gyros_rev_up high_speed_slew
        medium_speed_slew platform_damping_1 platform_damping_2
        playback{1} playback{2} position_tape{1} position_tape{2}
        position_tape{3} record_picture roll set_filter{1} set_filter{2}
        shutter_camera{1} shutter_camera{2} start_recording{1}
        start_recording{2} transmit_picture
        turn_off_camera turn_off_gyros turn_off_heaters
        turn_off_tape_recorder turn_on_camera turn_on_gyros
        turn_on_heaters turn_on_transmitter viewing{1} viewing{2}
        begin_earth_occultation end_earth_occultation
        datarate_change_1 datarate_change_2 datarate_change_3)

        (initact 0.0)
```

Figure 2: An abridged MIRRORS/II specification file for the self-processing network of the complete Spaceworld plan. Some details were omitted for brevity. (Page 1 of 3.)

```
        (attribute start dynamic optional)        ;--- Definition of node attributes needed for this model.
        (attribute duration dynamic optional)
        (attribute type dynamic)
        (method plan)                             ;--- Marker passing activation method for
                                                  ;--- non-hierarchical plans.
        (connects (plan oto incoming optional)]   ;--- Nodes in this set connect to other nodes in this set.

;--- The following node statements give detailed node attribute information for
;--- each node as well as describing the node connections in the network.
;--- Duration times and start times are given in seconds.

[implicit (member plan)]

[node calibrate_gyros (type action)]
[node change_data_mode{1} (start (0 nil 2255.76))(duration 300)(type action)
        (plan datamode_ok_3)]
                .
                .
                .
[node datamode_ok_3 (start (2405.76 nil 2555.76))(duration 0)(type inference)
        (plan transmit_picture)]
[node datamode_ok_4 (type inference)]
[node datamode_ok_5 (type inference)]
[node gyros_rev_up (type event)]
[node high_speed_slew (start (2405.76 nil 8801))(duration 99)(type action)
        (plan (/ platform_damping_1 viewing{2}))]
[node medium_speed_slew (start (0 nil 2370))(duration 80)(type action)
        (plan (/ platform_damping_2 viewing{1}))]
[node platform_damping_1 (start (2504.76 nil 8000))(duration 300)(type event)
        (plan shutter_camera{2})]
[node platform_damping_2 (start (80 nil 2450))(duration 100)(type event)
        (plan shutter_camera{1})]
[node playback{1} (start (20001 nil 22757.43))(duration 40)(type action)
        (plan (/ consolidate_tape{1} position_tape{3}))]
[node playback{2} (start (20170.29 nil 22926.71))(duration 73.28571)(type action)
        (plan (/ consolidate_tape{2} datarate_change_3))]
[node position_tape{1} (start (0 nil 7958.617))(duration 7.142857)(type action)
        (plan start_recording{1})]
                .
                .
                .
[node set_filter{1} (start (0 nil 2533.4))(duration 16.6)(type action)
        (plan shutter_camera{1})]
[node set_filter{2} (start (2405.76 nil 9176.5))(duration 23.5)(type action)
        (plan shutter_camera{2})]
[node shutter_camera{1} (start (2400 2500 2550))(duration 5.76)(type action)
        (plan (/ high_speed_slew set_filter{2} transmit_picture))]
[node shutter_camera{2} (start (9000 9000 9200))(duration 5.76)(type action)
        (plan record_picture)]
[node start_recording{1} (start (2948.76 nil 7965.76))(duration 1)(type action)
        (plan roll)]
[node start_recording{2} (start (9004.76 nil 9204.76))(duration 1)(type action)
        (plan record_picture)]
```

Figure 2: An abridged MIRRORS/II specification file for the self-processing network of the complete Spaceworld plan. Some details were omitted for brevity. (Page 2 of 3.)

```
[node transmit_picture (start (2405.76 nil 2555.76))(duration 144)(type event)
        (plan (/ change_data_mode{2} begin_earth_occultation))]
                .
                .
                .
[node turn_on_camera (start (0 nil 2520))(duration 30)(type action)
        (plan shutter_camera{1})]
[node turn_on_gyros (type action)]
[node turn_on_heaters (type action)]
[node turn_on_transmitter (start (0 nil 2255.76))(duration 300)(type action)
        (plan transmit_picture)]
[node viewing{1} (start (2400 nil 2550))(duration 0)(type inference)
        (plan shutter_camera{1})]
                .
                .
                .
[node datarate_change_1 (start (1000 nil 1000))(duration 300)(type event)
        (plan (/ viewing{1} datamode_ok_3 start_recording{1}))]
                .
                .
                .
;--- Control specification
[control ALTCONTROL]
[transcript plan]
[events (clamp)(show)]
;--- Start the nodes in the network which have no predecessor nodes.
[clamp (from 0 thru 1) (plan (&
        medium_speed_slew
        turn_on_camera
        set_filter{1}
        change_data_mode{1}
        turn_on_transmitter
        position_tape{1}))
        1.0]
[clamp (from 1000 thru 1001) (plan datarate_change_1) 1.0]
[run 25002]
[exit]
```

Figure 2: An abridged MIRRORS/II specification file for the self-processing network of the complete Spaceworld plan. Some details were omitted for brevity. (Page 3 of 3.)

A marker-passing paradigm was used as the spreading activation method for this self-processing network. Basically, a node representing a goal in a plan passes a marker to the nodes to which it sends outgoing connections, representing goals in the plan which are dependent on the completion of the sending node, when it has finished executing. Once a node which has not executed yet has received markers from all the nodes from which it receives incoming connections thereby satisfying the dependency constraint, and the starting time constraint has been satisfied, the node can begin executing. Nodes which do not have any dependency constraints can begin executing as soon as their starting time constraints are satisfied.

Using the network specification shown in Figure 2 and the marker-passing method described above the Spaceworld plan executed successfully, taking advantage of the parallelism inherent in the plan. The results of this execution can be seen in Figure 3. The output is composed of messages generated by nodes; each node prints a message to indicate when it began executing and when it stopped executing. Figure 3 shows that goals in the plan which are independent of each other are executed in parallel while other goals which must be executed in a specific order are executed sequentially. For example the nodes turn_on_camera and turn_on_transmitter both begin executing at the same time indicating that they are not dependent on each other and can be executed in parallel. Also, observe by comparing the network specification in Figure 2 to the output in Figure 3 that the starting time constraints of each goal have been satisfied and the duration time of each goal is accurate.

```
Beginning simulation, will stop at iteration 25002
Starting action change_data_mode{1} at time 0.
Starting action medium_speed_slew at time 0.
Starting action position_tape{1} at time 0.
Starting action set_filter{1} at time 0.
Starting action turn_on_camera at time 0.
Starting action turn_on_transmitter at time 0.
  Finished action position_tape{1} at time 7.
  Finished action set_filter{1} at time 16.
  Finished action turn_on_camera at time 30.
  Finished action medium_speed_slew at time 80.
Starting event platform_damping_2 at time 80.
  Finished event platform_damping_2 at time 180.
  Finished action change_data_mode{1} at time 300.
  Finished action turn_on_transmitter at time 300.
Starting event datarate_change_1 at time 1000.
  Finished event datarate_change_1 at time 1300.
Starting inference viewing{1} at time 2400.
  Finished inference viewing{1} at time 2400.
Starting inference datamode_ok_3 at time 2405.
  Finished inference datamode_ok_3 at time 2405.
Starting action shutter_camera{1} at time 2500.
  Finished action shutter_camera{1} at time 2505.
Starting action high_speed_slew at time 2505.
Starting action set_filter{2} at time 2505.
Starting event transmit_picture at time 2505.
  Finished action set_filter{2} at time 2528.
  Finished action high_speed_slew at time 2604.
Starting event platform_damping_1 at time 2604.
  Finished event transmit_picture at time 2649.
Starting action change_data_mode{2} at time 2649.
  Finished event platform_damping_1 at time 2904.
Starting action start_recording{1} at time 2948.
  Finished action change_data_mode{2} at time 2949.
  Finished action start_recording{1} at time 2949.
Starting action roll at time 2949.
  Finished action roll at time 3879.
Starting inference viewing{2} at time 3879.
  Finished inference viewing{2} at time 3879.
Starting action change_data_mode{3} at time 3888.
  Finished action change_data_mode{3} at time 4188.
Starting event begin_earth_occultation at time 5000.
  Finished event begin_earth_occultation at time 5001.
Starting event datarate_change_2 at time 7000.
  Finished event datarate_change_2 at time 7300.
Starting action shutter_camera{2} at time 9000.
```

Figure 3: Output of the self-processing network for non-hierarchical plan execution. (Page 1 of 2.)

```
Starting action start_recording{2} at time 9004.
  Finished action shutter_camera{2} at time 9005.
  Finished action start_recording{2} at time 9005.
Starting event record_picture at time 9005.
  Finished event record_picture at time 9045.
Starting action change_data_mode{4} at time 9053.
Starting action turn_off_tape_recorder at time 9053.
  Finished action turn_off_tape_recorder at time 9054.
Starting action position_tape{2} at time 9054.
  Finished action position_tape{2} at time 9094.
  Finished action change_data_mode{4} at time 9353.
Starting event end_earth_occultation at time 20000.
  Finished event end_earth_occultation at time 20001.
Starting action playback{1} at time 20001.
  Finished action playback{1} at time 20041.
Starting action consolidate_tape{1} at time 20049.
Starting action position_tape{3} at time 20049.
  Finished action consolidate_tape{1} at time 20049.
  Finished action position_tape{3} at time 20170.
Starting action playback{2} at time 20170.
  Finished action playback{2} at time 20243.
Starting action consolidate_tape{2} at time 20243.
  Finished action consolidate_tape{2} at time 20243.
Starting event datarate_change_3 at time 23000.
  Finished event datarate_change_3 at time 23300.
Starting inference clear_tape at time 25000.
  Finished inference clear_tape at time 25000.
```

Figure 3: Output of the self-processing network for non-hierarchical plan execution. (Page 2 of 2.)

V. Hierarchical Plan Execution

While non-hierarchical plans like that considered above order goals at a single level of abstraction, hierarchical plans consist of multiple levels of abstraction where each level "deeper" in the plan represents a more detailed level of abstraction. Based on the "lattice controller" described in [7] we were inspired to extend our research to include hierarchical plan execution. Hierarchical plans are generated by some AI planning systems so any general purpose plan execution scheme must be able to handle them. Hierarchical plans avoid some aspects of the computational complexity arising in real-world applications and are therefore of great value.

To develop a hierarchical plan to use for this research we added higher levels of abstraction to a subset of Vere's Spaceworld plan. The resulting plan can be seen in Figure 4. Each higher-level goal in the plan can be decomposed into more detailed goals. For example, the highest level goal in the plan is "Take a picture of the satellite Clotho" (take_picture_clotho). This goal can be broken down into the more detailed goals of "Photograph the satellite" (photograph_satellite) and "Transport the picture to earth" (transmit_picture_to_earth). Further levels of abstraction are illustrated in Figure 4. Goals in Figure 4 which have a numeric duration time are goals from the original non-hierarchical plan. The goals added to form the hierarchical plan do not have a specified duration time since their duration time is dependent on the starting and duration times of the goals one level lower in the hierarchy. Starting times of the higher-level goals were calculated based on the *earliest* early starting time and *earliest* latest starting time of goals of which the higher level goals are composed. For example, the early starting times of the nodes which are "children" of node photograph_satellite in the hierarchy are 0, 0, 0, 0, and 2400 so the early starting time of photograph_satellite node is 0 and the latest starting times of the children nodes are 2520, 2520, 2370, 1000, and 2550 so the latest starting time of the photograph_satellite node is 1000.

Nodes in the hierarchical planning network have two new node attributes, *parent* and *depend*, in addition to the start and duration attributes used in the non-hierarchical plan network. The parent attribute

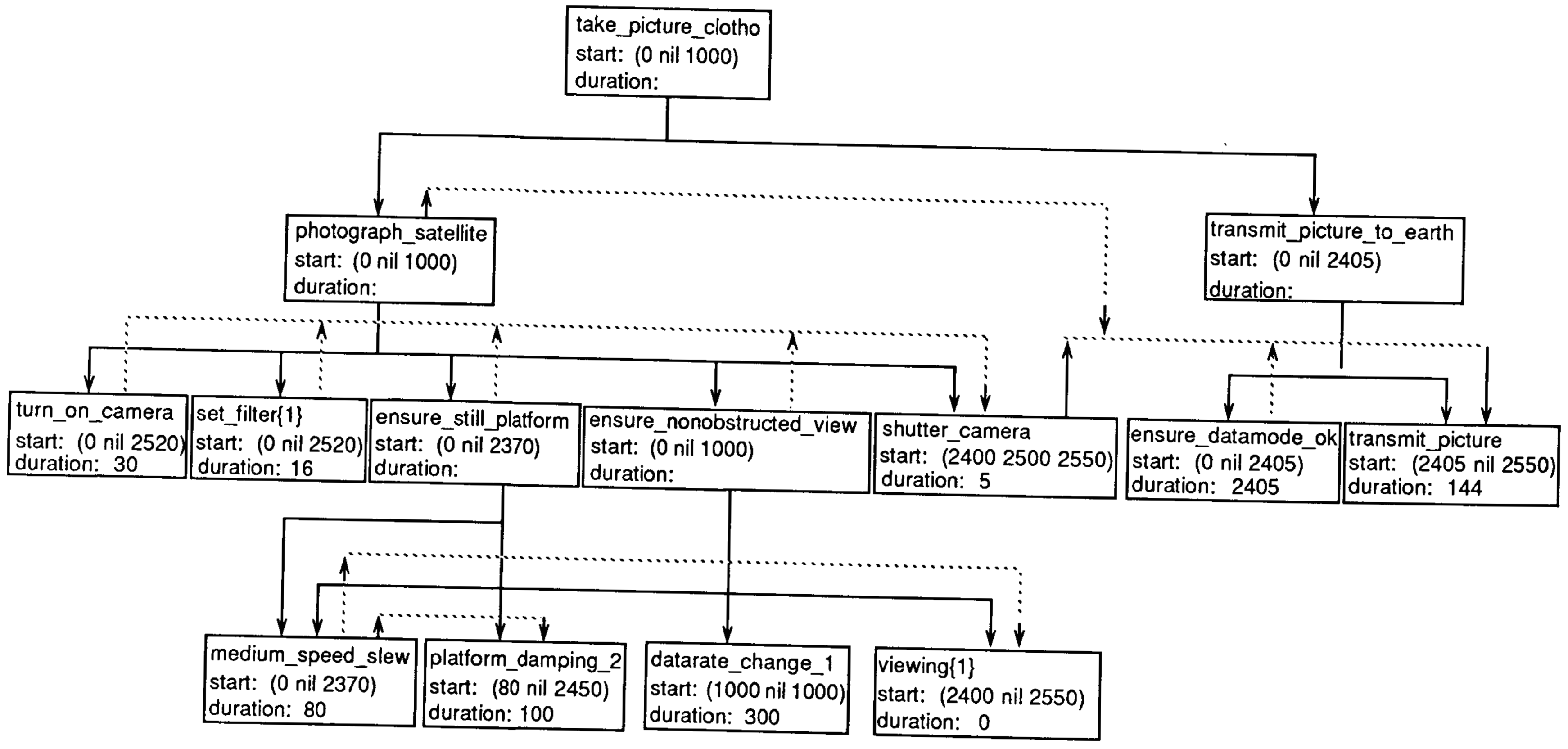

Figure 4: A hierarchical plan based on a portion of the Spaceworld Plan.

indicates what node or nodes are "above" a given node in the plan hierarchy (the solid lines in Figure 4). The depend attribute indicates nodes in the hierarchy which depend on the given node's execution completion to begin their own execution (the dotted lines in Figure 4). This order-dependent information is represented by node connections in both the non-hierarchical and hierarchical plan network. However, in the hierarchical plan, network node connections are also used to indicate more detailed goals which combine to represent the "parent" goal node. For example, the take_picture_clotho node connects to the photograph_satellite and transmit_picture_to_earth nodes since the latter two nodes represent a finer level of abstraction of the "parent" goal node take_picture_clotho (see Figure 5).

```
;--- Set containing the goals (nodes) in a hierarchical sebset of a Spaceworld plan.

[set plan (method hplan)              ;--- Activation method for hierarchical plans.
      (contains
            take_picture_clotho photograph_satellite
            transmit_picture_to_earth turn_on_camera set_filter{1}
            ensure_still_platform ensure_nonobstructed_view
            shutter_camera{1} ensure_datamode_ok transmit_picture
            medium_speed_slew platform_damping_2 datarate_change_1 viewing{1})
      (attribute start dynamic)            ;--- Definition of node attributes needed for this model.
      (attribute duration dynamic optional)
      (attribute parent dynamic optional)
      (attribute depend dynamic optional)
      (attribute type dynamic)
      (connects (plan oto incoming optional))]  ;--- Nodes in this set connect to other nodes in this set.

;--- The following node statements give detailed node attribute information
;--- for each node as well as describing the node connections in the network.

[implicit (member plan)]

[node take_picture_clotho (type action)(start (0 nil 1000))
      (plan (/ photograph_satellite transmit_picture_to_earth))]

[implicit (parent (take_picture_clotho))]
[node photograph_satellite (type action)(start (0 nil 1000))
      (depend (transmit_picture))
      (plan (/ turn_on_camera set_filter{1} ensure_still_platform
            ensure_nonobstructed_view shutter_camera{1} transmit_picture))]
[node transmit_picture_to_earth (type action)(start (0 nil 2405))
      (plan (/ ensure_datamode_ok transmit_picture))]

[implicit (parent (photograph_satellite))]
[node turn_on_camera (type action)(start (0 nil 2520))(duration 30)
      (depend (shutter_camera{1}))
      (plan shutter_camera{1})]
[node set_filter{1} (type action)(start (0 nil 2533))(duration 16)
      (depend (shutter_camera{1}))
      (plan shutter_camera{1})]
[node ensure_still_platform (type action)(start (0 nil 2370))
      (depend (shutter_camera{1}))
      (plan (/ medium_speed_slew platform_damping_2 shutter_camera{1}))]
```

Figure 5: A MIRRORS/II specification file for the self-processing network of the hierarchical plan shown in Figure 4. (Page 1 of 2.)

```
[node ensure_nonobstructed_view (type action)(start (0 nil 1000))
        (depend (shutter_camera{1}))
        (plan (/ medium_speed_slew datarate_change_1 viewing{1} shutter_camera{1}))]
[node shutter_camera{1} (type action)(start (2400 2500 2550))(duration 5)
        (depend (transmit_picture))
        (plan transmit_picture)]

[implicit (parent (transmit_picture_to_earth))]
[node ensure_datamode_ok (type action)(start (0 nil 2405))(duration 2405)
        (depend (transmit_picture))
        (plan transmit_picture)]
[node transmit_picture (type action)(start (2405 nil 2555))(duration 144)]

[node medium_speed_slew (type action) (start (0 nil 2370))(duration 80)
        (parent (ensure_still_platform ensure_nonobstructed_view))
        (depend (platform_damping_2 viewing{1}))
        (plan (/ platform_damping_2 viewing{1}))]
[node platform_damping_2 (type action)
        (start (80 nil 2450))(duration 100)
        (parent (ensure_still_platform))]

[implicit (parent (ensure_nonobstructed_view))]
[node datarate_change_1 (type event)(start (1000 nil 1000))(duration 300)
        (depend (viewing{1}))(plan viewing{1})]
[node viewing{1} (type inference)(start (2400 nil 2550))(duration 0)]

;-------- control spec
[events (input)(show)]
[control ALTCONTROL]
;--- Start the top-level node in the hierarchy
[input (from 0 thru 1)(plan take_picture_clotho) 1.0]
[transcript hplan]
[run 2655]
[exit]
```

Figure 5: A MIRRORS/II specification file for the self-processing network of the hierarchical plan shown in Figure 4. (Page 2 of 2.)

The marker passing algorithm used for hierarchical plan execution differs slightly from the one used for non-hierarchical plan execution. In hierarchical plan execution, goal execution begins with the node at the top-most level of abstraction. Once a parent node begins executing it sends a marker to each of its "child" nodes. Each "child" node must receive a marker from each node on which it depends <u>and</u> from it's parent node before it can begin executing; it is not appropriate to begin executing nodes at lower levels of the hierarchy if the conditions for executing their parent goals at higher levels of the hierarchy have not been met. Once a "child" node has completed executing, it sends a marker to its "parent" node indicating that it is done and to any other nodes which depend on the completion of its execution. All the child nodes (those one level lower in the hierarchy) must complete executing before the parent node's execution can be considered complete. The starting time constraints remain the same.

Using the self-processing network shown in Figure 5 and the marker-passing method described above, the hierarchical plan executed successfully. The results are shown in Figure 6. You can see that the top-most node began executing first and finished executing last because it could not complete executing until all its lower-level detail nodes had finished executing. Many of the goals were executed in parallel. Also note that the necessary sequential processing was maintained. For example, the transmit_picture goal did not begin executing until the photograph_satellite node was finished executing.

```
       Beginning simulation, will stop at iteration 2655
       Starting action take_picture_clotho at time 0.
       Starting action photograph_satellite at time 0.
       Starting action transmit_picture_to_earth at time 0.
       Starting action turn_on_camera at time 0.
       Starting action set_filter{1} at time 0.
       Starting action ensure_still_platform at time 0.
       Starting action ensure_nonobstructed_view at time 0.
       Starting action ensure_datamode_ok at time 0.
       Starting action medium_speed_slew at time 0.
         Finished action set_filter{1} at time 16.

      Finished action turn_on_camera at time 30.
      Finished action medium_speed_slew at time 80.
    Starting action platform_damping_2 at time 80.
      Finished action platform_damping_2 at time 180.
      Finished action ensure_still_platform at time 181.
    Starting event datarate_change_1 at time 1000.
      Finished event datarate_change_1 at time 1300.
    Starting inference viewing{1} at time 2400.
      Finished inference viewing{1} at time 2400.
      Finished action ensure_nonobstructed_view at time 2402.
      Finished action ensure_datamode_ok at time 2405.
    Starting action shutter_camera{1} at time 2500.
      Finished action shutter_camera{1} at time 2505.
      Finished action photograph_satellite at time 2506.
    Starting action transmit_picture at time 2506.
      Finished action transmit_picture at time 2650.
      Finished action transmit_picture_to_earth at time 2651.
      Finished action take_picture_clotho at time 2652.
```

Figure 6: Output of the self-processing network for hierarchical plan execution.

VI. Discussion

Along with recent related work [7], this study demonstrates for the first time that plan execution, a task usually solved using traditional AI problem-solving techniques, can be accomplished using a self-processing network. The distributed processing approach used here allows many plan steps to be executed in parallel while still preserving the essential sequential aspects of the plan execution. The fact that self-processing networks have been applied to various space-related applications [5,6] in addition to the one discussed here demonstrates the general aplicability of this approach to planning and control problems relevant to spacecraft activities. A logical next step might be to implement a self-processing model which could execute plans which are dynamically changing during the period of execution.

This work also demonstrates that MIRRORS/II is a powerful environment for the development/evaluation of self-processing systems in general. It allowed us to develop this plan execution model in a very short amount of time and to implement marker passing processing paradigms as needed. The design of MIRRORS/II and all previous work with MIRRORS/II had been limited to connectionist models and had not considered the possibility of using methods like marker passing. The ease with which MIRRORS/II supported marker passing methods without any alterations to MIRRORS/II itself suggests that it will prove quite robust as a software environment for future automation research. A logical next step might be the development of other parallel AI methods in the context of MIRRORS/II.

Acknowledgements: Supported in part by NASA award NAG1-885 and in part by NSF award IRI-8451430.

VII. References

[1] D'Autrechy, C.L., et al., A General-Purpose Environment for Developing Connectionist Models, *Simulation*, 51, 1988, 5-19.

[2] D'Autrechy, C.L., et al., *MIRRORS/II Reference Manual*, 1988.

[3] Reggia, J. & Sutton, G., Self-Processing Networks and Their Biomedical Implications, *Proc. of the IEEE*, 76, 1988, 680-692.

[4] Vere, S., Planning in Time: Windows and Durations for Activities and Goals. *IEEE Trans. Pat. Anal. & Mach. Intel.*, 1988.

[5] Whitfield, K., et al., A Competition-Based Connectionist Model for Dynamic Control, in this Proceedings, 1989.

[6] Peng, Y. and Reggia, J., A Connectionist Model for Diagnostic Problem Solving, *IEEE Trans. Sys., Man and Cyber.*, 1989, in press.

[7] Sliwa, N. and Soloway, D., A Lattice Controller for Telerobotic Systems, *American Controls Conference*, 1987.