Journal of Intelligent and Robotic Systems 9: 209–233, 1994.

Coordination Specification for CIRSSE Robotic Platform System Using Petri Net Transducers

FEI-YUE WANG
Department of Systems and Industrial Engineering, University of Arizona, Tucson, AZ 85721, U.S.A.

MICHAEL MITTMANN, and GEORGE N. SARIDIS
NASA Center for Intelligent Robotic Systems for Space Exploration, Rensselaer Polytechnic Institute, Troy, NY 12180, U.S.A.

(Received: 5 December 1991; revised: 22 September 1992)

Abstract. A formal model based on Petri net transducers has been developed to specify the coordination and communication among the various task units in the CIRSSE platform system for robotic construction in space stations. The specification guarantees a mechanism of coherent control and communication for the effective cooperation among the different task units, and outlines the major steps toward the integration of the robotic platform system. The model is based on a coordination structure consisting of one dispatcher and three coordinators representing, respectively, the motion, vision, and gripper units of the platform system. The coordination structure insures some desired process properties for the system, such as boundedness, liveness, and reversibility, and easier translation from the formal specifications to the program codes based on Petri net transducer models. The model also assists with the system development in many ways, including (i) reducing the number of errors introduced while converting specifications to codes; (ii) assisting the developers in program implementation and verification (iii) allowing quicker adaptation to changed specifications; and (iv) allowing easier testing of the results for specification modifications. Therefore, it provides a useful tool for the design, simulation, performance evaluation, and implementation verification of the CIRSSE robotic platform system.

Key words. Petri net transducer, coordination structure, command language, communication, intelligent machines.

1. Introduction

A platform system for robotic construction in space stations is currently under development at the NASA Center for Intelligent Robotic Systems for Space Exploration of Rensselaer Polytechnic Institute. The system consists of a platform with two PUMA manipulators mounted on two moving bases. Each manipulator has six degrees of freedom of motion and each base has three degrees of freedom. A vision system is incorporated into the platform system for object identification and location determination. The task scenario for the platform system is to assemble strut structures in a dynamic and uncertain environment on the space stations (Figure 1).

The effort for the design and implementation of such systems is enormous. To carry out the construction mission successfully, the CIRSSE platform system has to cope with a variety of environment conditions. Moving entities, such as other arms, mobile robots, parts, *etc.*, may appear in the workspace of the platform system in

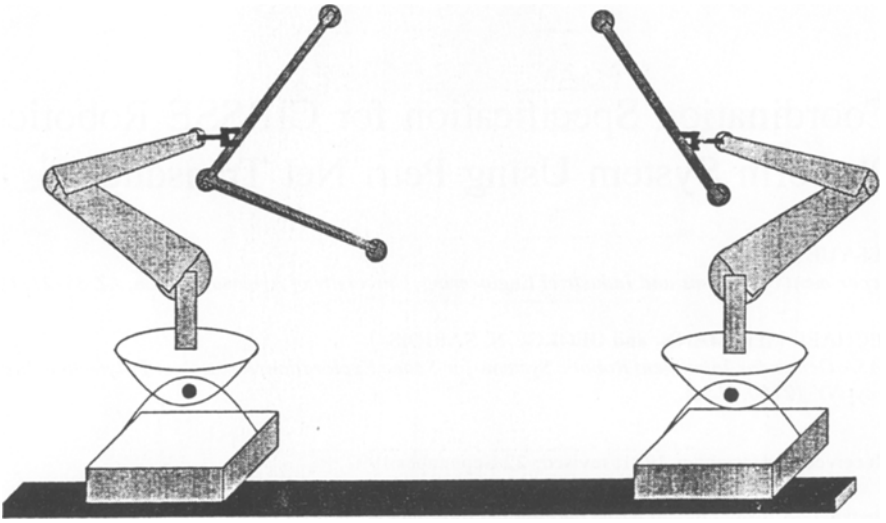

Fig. 1. The CIRSSE robotic platform system.

some unpredictable ways. Another major difficulty faced by the system is the time delay and information reliability caused by the communication between the earth stations and the space stations. Since it is undesired and infeasible to have human operators to maintain the operation of the system on space stations, it is imperative to design an autonomous intelligent control system for the platform system to perform all system operations with minimum assistance from human operators.

The intelligent control system for the CIRSSE robotic platform will be developed according to the theory of *Hierarchical Intelligent Control Systems* proposed by Saridis (Saridis, 1977). Based on this theory, the control system architecture is arranged into three levels: the *Organization, Coordination,* and *Execution Levels,* hierarchically ordered according to the principle of *Increasing Precision with Decreasing Intelligence* (Figure 2). Since the late 1970s, significant effort has been made by Saridis and his colleagues towards the establishment of general mathematical models for these levels and their corresponding integration problems (Saridis and Stephanou, 1977; Saridis, 1985; Saridis and Valavanis, 1988; Valavanis and Saridis, 1991; Wang and Saridis, 1990, 1991, 1992; Wang *et al.*, 1991). For the platform system, the function of the Organization Level is to define construction missions and to generate the high level task plans for some specific strut structures. The plans include the specification of structure configuration, strut/node assembling sequence, and motion task commands. The Coordination Level serves as an interface between the Organization and Execution levels. Its main function is to translate the higher level task plans into the specific operation instructions and then coordinate their execution. Finally, the Execution Level is to execute the operation instructions using control methods through various devices.

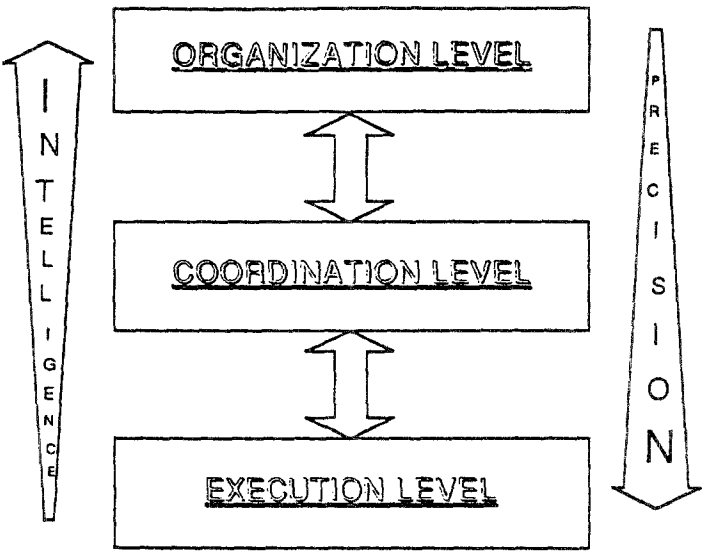

Fig. 2. The hierarchical structure of intelligent control systems.

The focus of this paper is on the specification and implementation of the Coordination Level of the CIRSSE robotic platform system. The key issue here is to specify a *mechanism of coherent control and communication* for the diversified task units such as motion and vision subsystems in the platform system, which will guarantee the effective cooperation among the task units and the continuous operation of the system toward a successful construction mission.

The specification stage is perhaps the most critical step in system development. It has been estimated that the cost of reworking an error discovered in the coding phase of a project is 50 to 200 times more expensive than fixing an error discovered during specification (Boehm, 1976). In addition to reducing the cost and development time of a system, good specification will insure that the system does the desired task, and allow better estimates of the functionality and the time to produce the system. For system implementation, when an appropriate formalism has been chosen for specification, system specifications can be automatically translated into executable codes (Krogh *et al.*, 1988).

For the CIRSSE platform system, the coordination structures developed for Intelligent Machines by Wang and Saridis (Wang and Saridis, 1990) have been used for the specification of its Coordination Level. The advantages of using the coordination structures include the guarantee of the desired system process properties, such as boundedness, liveness, and reversibility, and easier translation from the formal specifications to program codes based on Petri net transducer (PNT) models. More specific, a coordination structure consisting of one dispatcher and three coordinators has been constructed for the CIRSSE platform system. The dispatcher will receive the compiled task plans from the higher level of the system and deal

with the control and communication of the coordinators during the execution of the task plans. The control and communication is achieved by translating the given task plan into a sequence of coordinator-oriented control actions, and dispatching them to the corresponding coordinators. The three coordinators will supervise the operation and data passing in the motion, vision, and gripper subsystems of the platform, respectively. Each coordinator will translate the given sequence of control actions into a sequence of executable operation instructions, and send them to the involved devices. Both dispatcher and coordinators are modeled as PNTs and implemented on different computers which host the corresponding subsystems.

The coordination structure defines formally the overall architecture and information flow for the Coordination Level of the CIRSSE platform system. Besides the above mentioned process properties guaranteed for the system by the coordination structure, it also assists with the system development in many other ways, including

- Reducing the number of errors introduced while converting specifications to codes.
- Assisting the developers in program implementation and verification.
- Allowing quicker adaptation to changed specifications.
- Allowing easier testing of the results of specification modifications.

Therefore, the coordination structure provides a useful tool for the design, simulation, performance evaluation, and implementation verification of the CIRSSE platform system. Note that applications of Petri nets for specification and verification of other kinds of systems have also been investigated (Azema *et al.*, 1984; Bruno and Marchetto, 1986; Courvoisier *et al.*, 1986).

Section 2 presents a brief overview of the CIRSSE platform system project. Section 3 gives the system command language and task grammar for expressing input task plans to the Coordination Level of the system. Section 4 specifies in detail the PNT models for the dispatcher and coordinators with the corresponding pseudo-codes. Implementation and communication mechanism are described in Section 5. Simulation results of testing the speed of communication in various situations are also discussed in this section. Finally, Section 6 concludes the paper with a summary and suggestions for future research.

2. Overview of the CIRSSE Platform System Project

The present goal of the CIRSSE platform system project is to be able to assemble struts and nodes autonomously onto some well-defined structures in a typical space environment. To simplify the problem, it has been assumed that there are only three classes of objects in the workspace of the platform: struts, nodes, and obstacles. The obstacles may travel in the workspace in some unpredicatable ways, and the platform system has to avoid the possible collision with them during the construction process.

The high level task plans for construction will be generated by the Organization Level sited on the earth station and transmitted to the Coordination Level hosted on the space station. To interact with the Coordination Level, a formalism, called *system command language,* has to be designed to express the construction missions. The actual input to the Coordination Level will be compiled system commands (or task plans). Obviously, to reduce the communication intensity between the earth station and the space station, the compiler for the system command language should be hosted on the space station along with the Coordination Level of the platform system.

Due to the constraint imposed by the earth-space communication, the Coordination Level has to be able to operate under the situation that task commands from the high level arrive infrequently and erratically. To accomplish this, the dispatcher (D) of the Coordination Level should

(1) Decompose the task commands into subtasks and dispatch them in order when the relevant coordinators have signaled that they are ready for the next execution.

(2) Communicate with the coordinators relatively fast.

(3) Be capable of setting up communication between any two coordinators which need information exchange.

There are three coordinators in this Level: the vision coordinator (VC), the motion coordinator (MC), and the gripper coordinator (GC). The system components of the dispatcher and the coordinators are described in the sequel.

2.1. THE DISPATCHER (D)

As noted above, the communication time from the high level to the dispatcher is expected to be erratic, however, it is required that the dispatcher communicates quickly with any of the coordinators. For a given task plan, the dispatcher is concerned primarily with questions related to which coordinator(s) should be called for tasks (*task sharing*) and/or be informed by the status of the task execution (*result sharing*). The dispatcher is not involved with the actual transmission of large blocks of data. If large amounts of data is expected to be passed from one coordinator to another, then the dispatcher will just instruct the coordinators to connect to each other, and allow them to communicate at whatever rate they are capable of.

The dispatcher is physically realized on a SUN 4/260 workstation. The dispatcher communicates with the coordinators through THINNET, a local version of ETHERNET. The communication is implemented with IPC SOCK_STREAM type sockets under the PF_INET protocol. This implementation results in a minimum turn around time of 200 ms from one processor to another and back, which is within the limits of our design specifications.

2.2. THE VISION COORDINATOR (VC)

The vision coordinator has the job of 'looking' at an area that it is told to observe, attempting to find something it has been told to find there. It then gives the location of the identified object back to the system which asked for it, and waits for another command. The location of an object includes its position and orientation, and in the case of obstacles, a simple description of the obstacle geometry.

The vision system is physically realized on a VxWorks cage with a Data cube and a Motorola MVME147 controller. The controller is connected to the THINNET, and thus to the rest of the system. The Datacube boards consist of:

SNAP. The SNAP board performs non-linear transformations (comparisons and max/min determinations) is sequential digital video data.

VFIR-MKIII. A video impulse response filter module. It implements a 256 arbitrary coefficient convolution. This is primarily used in edge detection and noise filtering.

MAX-SP. This is capable of performing real time frame rate single point temporal and spatial filters, image merging, image subtraction and addition, and Min/Max processing.

FEATURE MAX-MKII. This does advanced feature-list extraction, and histograms. A summation of all row or column pixels can be done in a table.

MAX-MUX. This provides the MaxVideo user with software control over MAXbus data source and destination selection. This allows for easier reprogramming of the Datacube.

DIGIMAX. This is a video acquisition and display module which is capable of accepting one of eight inputs. This is used to feed the information from the cameras to the ROI-STORE units.

ROI-STORE. This is a frame storage module which supports user programmable video resolution and processing of regions of interest within a video image.

Since there are two DIGIMAX cards, two frames can be read simultaneously. There are five cameras in the vision system. Two of the cameras are mounted on the manipulators, allowing greater diversity in the objects which can be viewed, but presenting greater challenges in calibration. Two of the remaining cameras are mounted on the ceiling of the workspace, and the remaining one has not yet been placed. Each frame grabber is capable of reading the cameras at a rate of 30 Hz. The output of the Datacube is sent to the MVME147 which can further analyze the image and send data to other coordinators. The MVME147 does all intermediate level vision operations, such as Hough transforms and line fitting.

The Uniphase laser is controlled by the Motorola MVME 135 CPU and the MVME 340 parallel board. The laser is used to put bright points on the object and to make stereo point matching easier.

2.3. THE MOTION COORDINATOR (MC)

The motion coordinator is used to control the robot arms to move objects in the environment, and to move the cameras which are attached to the arms. The coordinator also participates directly in the camera calibration with the vision coordinator.

The motion system is physically realized on a VxWorks cage running 5 Motorola MVME 135 boards (68020 CPUs), along with

MVME 340A. A parallel port board, which also supplies timer interrupts. This is used to read the sensors, and supply interrupts for the platform servo control.

VMIVME 2532A. A digital I/O board, which is used primarily for switching external circuits and thus allowing software control of power to any of the arms.

DVME 628. A D/A converter for supplying motor currents to the arms. The digital signal is converted to analog and then run through a servo amplifier and fed to the joints in the system.

MVME 224-1. Four MBytes of shared memory.

XVME 556. A 16 channel A/D converter which is currently unused, but may be used for reading encorders.

Whetdco Encoder. A VME 3570 optical shaft encoder used for reading the position of the carts on the Aronson platform.

VME 7016. A VME Q Bus controller used to control the PUMAs.

332 XT. Eight channel serial interface. This will be used to control the grippers.

The 68020s connect via the databus to a D/A board which feeds currents to a PUMA 560 and a PUMA 600 arms. The PUMA arms have absolute position potentiometers, and torque sensors at all of their joints. Each arm is mounted on an Aronson platform which gives each arm three more degrees of freedom for motion. The arms and platform are controlled by *Kali* (Topper *et al.*, 1988), an integrated path planner/arm controller.

2.4. THE GRIPPER COORDINATOR (GC)

The grippers will be used to actually grasp struts and nodes. They must be able to sense when there is an object between their 'fingers' and report the finger position and the force they are applying. The grippers are pneumatically controlled ones. Each gripper is equipped with a crossfire sensor and a force sensor, and is mounted on a Lord force/torque sensor, which in turen is mounted on the end of the PUMA arm. The gripper controller is a Motorola 68HC11 based controller, which communicates witht the VxWorks cage through the 332 XT serial interface in the VxWorks cage.

3. System Command Language and Task Grammar

As mentioned in the previous section, a system command language is necessary for the Coordination Level to interact with the Organization Level. This high level

command language also makes the programming of the construction missions easier for human operators.

A general formalism for the system command language can be defined by following the syntaxs of the existing high level languages, such as Pascal or VAL-II. One of such examples is the command formalism developed by Noreils and Chatila (1989) for a mobile robot system. However, since this paper is concentrated only on the Coordination Level of the platform system and the inputs to that Level are the compiled system commands only (that is, sequences of tasks which are directly related to the operation of this Level), no attempt is made here to specify a formal language for system commands. Instead, we define the following *task grammar G* to describe the compiled system commands to the Coordination Level:

$$G = (S, N, \Sigma_0, P)$$

where

$$\Sigma_0 = \{cal\_r, cal\_v, move, approach, release, grasp, find\_sn, find\_obs, slave, continue\_v\},$$

$$N = \{S, V, V_t, M, M_s, M_v, H_s, H_{s1}, H_{s2}, H_v, H_{v1}, H_{v2}\},$$

$$P = \{S \rightarrow cal\_rM, M \rightarrow cal\_vV | M_s, M_s \rightarrow move\ M_s | approach\ M_s | approach\ H_s, H_s \rightarrow release\ H_{s1}, H_{s1} \rightarrow grasp\ H_{s2} | move\ H_{s1} | approach\ H_{s1} | cal\_vV, H_{s2} \rightarrow H_v | M_s, V \rightarrow find\_snV | find\_obsV | slave\ V_t | M_v, V_t \rightarrow continue\_vV, M_v \rightarrow move\ M_v | approach\ M_v | approach\ H_v | V, H_v \rightarrow release\ H_{v1}, H_{v1} \rightarrow grasp\ H_{v2} | move\ H_{v1} | approach\ H_{v1} | V, H_{v2} \rightarrow H_v | M_v, V, M, M_s, M_v, H_s, H_{s1}, H_v, H_{v1} \rightarrow S\}.$$

Σ_o represents the set of task primitives (terminal symbols), N is the set of non-terminal symbols with S as the start symbol, and P the set of production rules for deriving task plans (or language). The meaning of these task primitives will become clear later in the next section.

The task grammar G characterizes the basic task precedences in the operation of the platform system. With the task grammar, the problem of developing the Coordination Level becomes that of constructing a coordination structure which is capable of processing all the task plans generated by G. This will be accomplished in the following section by giving the individual PNTs for the dispatcher and coordinators.

4. Coordination Structure for the Platform System

This section describes in detail the PNT models for the dispatcher and coordinators. A PNT is a language translator, defined by a 6-tuple $(N, \Sigma, \Delta, \sigma, \mu, F)$, which trans-

lates a given input task plan into an output task plan. The Petri net $N = (P, T, I, O)$ of PNT is the controller of translation, and μ is the *initial marking* of N, i.e., the initial state of PNT. Σ is the *input alphabet* representing input tasks, and Δ is the *output alphabet* representing the output tasks. σ, the *translation mapping* from $T \times (\Sigma \cup \{\lambda\})$ to finite sets of Δ^* (where λ is the empty string and Δ^* is the set of all strings of finite length over Δ), specifies for a given input task the processing transitions in N as well as the output sub-task plans which may be used for that task. Finally, F is a set of *final markings* indicating the termination of the task translation. The translation mappings for the dispatcher and coordinators are specified by pseudo C codes here. For details of PNs, PNTs, coordination structures, and their properties, a reader is referred to Wang and Saridis (1990, 1991 and 1992).

4.1. PNT FOR THE DISPATCHER

The Petri net model for the dispatcher is given in Figure 3(a). For the sake of simplicity, however, a simplified net, shown in Figure 3(b), for the dispatcher will be discussed here. The simplified net is obtained by replacing some subnets of the original net with so-called macroplaces/transitions without changing

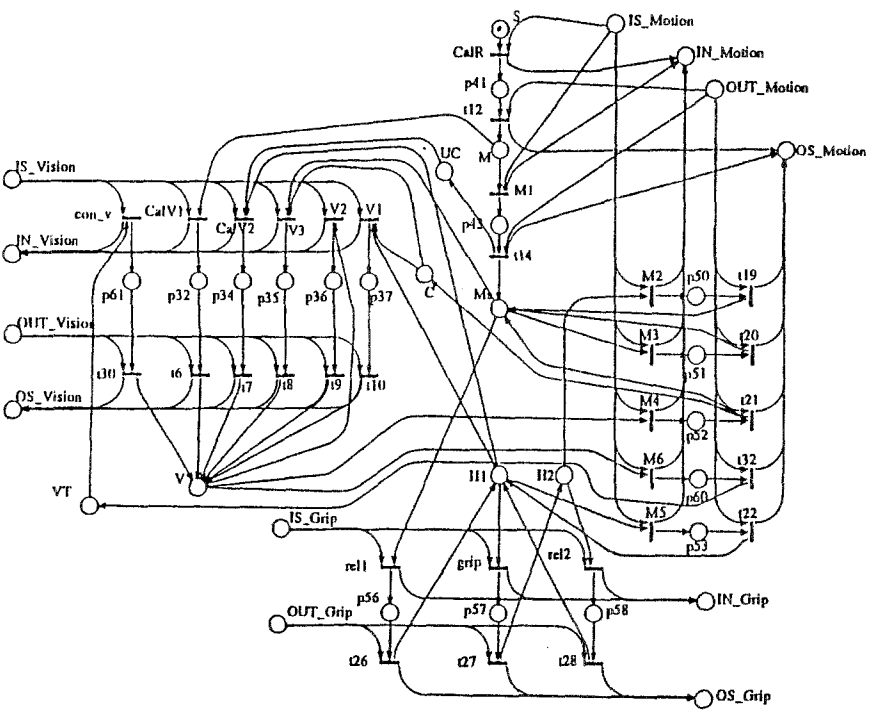

Fig. 3a. The Petri net transducer for dispatcher.

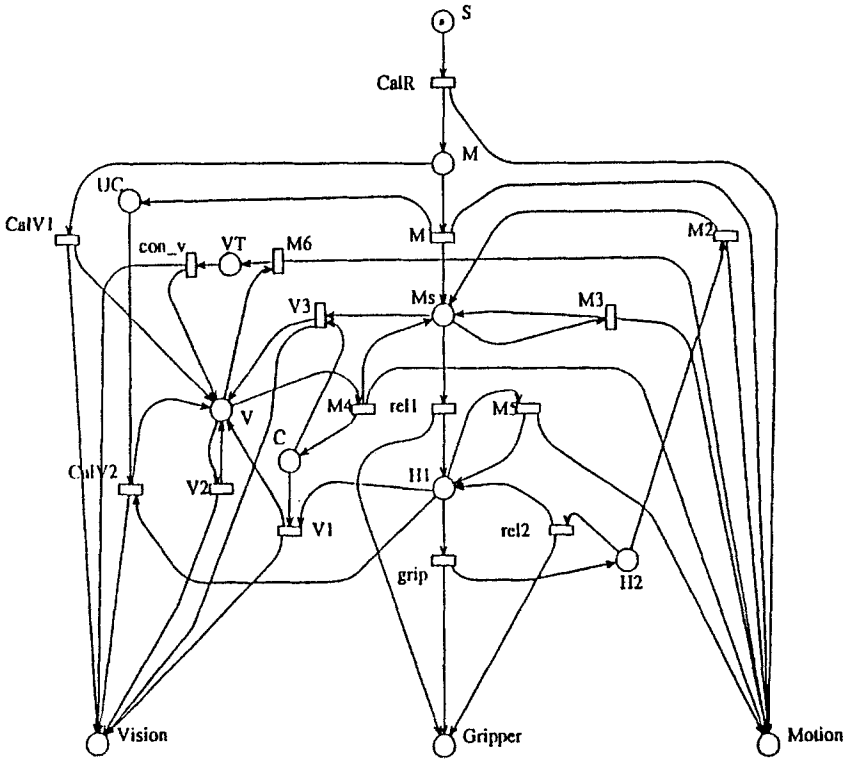

Fig. 3b. The simplified Petri net transducer for dispatcher.

the net topology, and thus knowing the simplified one would be enough for understanding the task processes of the dispatcher. The simplified Petri net consists of 12 places and 16 transitions. A transition generally represents dispatching a command to the coordinators for performing a specific task, while a place represents the state of the system. These places and transitions are specified as follows:

Transitions:

CalR: Send a *calibrate* command to MC and VC.

M1, M2, M3, M4, M5: Send a *move* or *approach* command to MC.

Grip: Send a *grasp* command to GC.

Rel1, Rel2: Send a *release* command to GC.

V1, V2, V3: Send a *find obstacle, find strut,* or *find node* command to VC.

CalV1, CalV2: Send a *calibrate* command to VC.

ConV: Send a *continue* command to the interrupted task of VC.

Places:

C: Holding a token if the cameras have been calibrated.

UC: Holding a token if the cameras are not calibrated and at least one move command has been done.

S, M, Ms, H1, H2, V, Vt: Places correspond to the nonterminals in grammar G.

Vision, Gripper, Motion: Each of these three places represent four input, output, and semaphores for each of the three subnets (Figures 4–6)

The input alphabet for D, i.e., the tasks to be translated by D, is the set of primitive tasks specified by Σ_o as in grammar G. The output alphabet, i.e., the control commands to be dispatched by D to the coordinators, is $\Delta_d = \Sigma_v \cup \Sigma_m \cup \Sigma_g$, where Σ_v, Σ_g, and Σ_m are the input alphabets for VC, MC, and GC, respectively. We have,

$$\Sigma_v = \{look, find, calibrate, continue\},$$

$$\Sigma_m = \{approaches, move\},$$

$$\Sigma_g = \{crossfire, goto\_position, goto\_force, read\_position, read\_force\}.$$

The translation mapping $\sigma_d(t, s)$ of D specifies the procedures of processing task s by transition t. This is achieved by translating each of tasks in Σ_o into task strings over Δ_d. For example,

$$\sigma_d(Grip, grasp) = \{((goto\_position.read\_position)^+ crossfire)^+.goto\_force\},$$

where $s^+ = \{s^n, n \geqslant 1\}$ represents repetitive actions. Among all possible task strings for *grasp*, only one will be selected for each translation and the selected string will be sent to *GC* via communication for execution.

Other translations in D are: $\sigma_d(V, find\_x)$, where V = V1, V2, or V3, and $find\_x = find\_sn$ or $find\_obs$; $\sigma_d(Rel, release)$, Rel = Rel1 or Rel2; $\sigma_d(CalV, cal\_v)$, $CalV$ = CalV1 or CalV2; $\sigma_d(ConV, continue\_v)$; $\sigma_d(CalR, cal\_r)$; $\sigma_d(M, x)$, M = M1, M2, M3, M4, M5, or M6, and x = *approach* or *move*; $\sigma_d(M6, slave)$. All transitions associating with no input tasks are *internal* operations, which can be fired without processing any external tasks. It is easy to verify that all input task plans generated by grammar G can be translated by the dispatcher described.

4.2. PNT FOR THE VISION COORDINATOR

The Petri net model of the vision coordinator in Figure 4 consists of 15 places and 16 transitions. The transitions generally represent lower level routines for task execution, while the places represent the state of the vision system. The places and transitions are specified as follows:

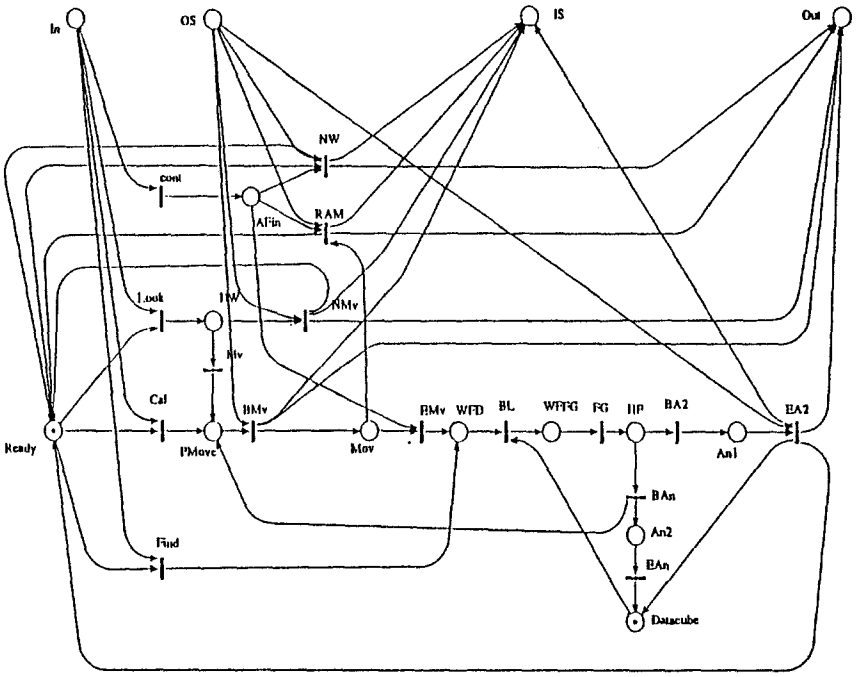

Fig. 4. The Petri net transducer for vision coordinator.

Transitions:

Look: Position the vision system to 'look' at a given location.

NMv: Require no arm movement to 'look' at a given location.

Mv: Require arm movement to 'look' at a given location.

BMv: Send the motion command to MC.

Cal: Execute the given calibration command.

Find: Give the command to identify a strut, node, or an obstacle that the vision system is already looking at.

EMv: Receive a non-error message from MC.

BL: Block until the Datacube is free.

FG: Get pictures from the cameras through frame grabber, and send data to the Datacube.

BAn: Begin partial image processing in the case when another motion command must be sent to MC to take new pictures.

EAn: Reset the Datacube be ready when partial image processing is completed.

BA2: Begin the final image processing.

EA2: Send the results of image analysis to the Out place and reset VC to be ready.

RAM: Return from Arm Movement, this is fired if the arm returns an error, or if the *look* task is being done.

Cont: Fire when the event VC is waiting on (e.g., an arm motion) is completed.

NW: Fire if VC receives a *continue* command when it isn't waiting.

Places:

In, Out: the input and output places.

IS, OS: The input and output semaphores.

Ready: Marking the availability of the system.

Datacube: Marking the availability of the Datacube.

An1, An2: the Datacube is processing information.

Mov: The arm is moving.

IA, OA: Input and Output to the Arm.

PMove: Preparing to Move the arm

AFin: The arm motion is completed.

WFD: Waiting For Datacube.

HF: Having Frame.

RMov: Ready to move the arm.

DW: Deciding where cameras need to be in order to 'look' at the given location.

WFFG: Waiting For Frame Grab.

The set of subtasks to be processed by the vision coordinator is Σ_v, are given in the previous subsection. The output alphabet Δ_v consists of procedures for controlling camera devices, processing digital images, communicating with the motion coordinator. The translation mapping σ_v of VC specifies the association of input tasks, transitions, and execution procedures. For example, translation $\sigma_v(Look, look) = \{select\_cameras.position\_cameras, select\_cameras.calculate\_arm\_position.position\_cameras\}$ indicates that transition Look can execute task *look* by using either $select\_cameras.position\_cameras$ or $select\_cameras.calculate\_arm\_position.position\_cameras$. In the first case, the task can be performed by selecting two cameras among all available ones and then positioning the selected cameras, while in the second case arms have to be moved in order to position the cameras at the desired location. The three routines are specified in the sequel:

```
select_cameras(x,y,z,cameras_allowed)
      float x,y,z;                    /* the location to be looked at    */
      boolean cameras_allowed[];      /* specifies all available cameras */
      {
      int i;
      for(i=0; i< NUM_CAMERAS; i++)
            cameras_allowed[i] &= can_be_pointed_at(x,y,z, i);
      if (number_of_allowed(cameras_allowed) < 2) return (ERROR);


      else enable_two_best_cameras(cameras_allowed);
      return(enabled_camera_descriptor);
      }

calculate_arm_position(x,y,z,cameras_enabled)
      float x,y,z;                    /* the location to be looked at */
      boolean cameras_enabled[];      /* specifies selected cameras   */
      {
      if(number_of_cameras_mobile(cameras_enabled) != 1)
            return(ERROR);
      else load (where_the_camera_should_be(x,y,z),DESIRED_ARM_POSITION_TABLE);
      return(OK);
      }

position_cameras(x,y,z,cameras_enabled) /* procedure of positioning cameras */
```

Translation $\sigma_v(Cal, calibrate)$ is performed by,

```
load_desired_arm_positions_for_calibration();
analyze = &analyze_calibration_card_routine;
```

Translation $\sigma_v(Find, find)$ is performed by,

```
setup_analyze_routine(tape_cmnd, high_precision)
boolean high_precision;
int tape_cmnd;
{
if(high_precision){
      switch(tape_cmnd) of
         case    strut:      analyze_routine = &find_strut_routine_hp;
         case    node :      analyze_routine = &find_node_routine_hp;
         case    obstacle:   analyze_routine = &find_obstacle_routine_hp;
          }
       }
```

```
else
      switch(tape_cmnd) of
         case    strut:      analyze_routine = &find_strut_routine;

         case    node :      analyze_routine = &find_node_routine;
         case    obstacle:   analyze_routine = &find_obstacle_routine;
          }
      }
}
```

The final translation, $\sigma_v(Cont, continue)$, just informs VC that the arm movement it is waiting on is completed, and VC can continue its processing. All other transitions are internal operations of VC.

4.3. PNT FOR THE MOTION COORDINATOR

There are 13 places and 9 transitions in the Petri net model of the motion coordinator (Figure 5). The meanings of places and transitions are specified as follows:

Transitions:

St: Start the motion task.

Ill: Report an illegal location for the arm to go to.

Calc: Calculate the transform for the Cartesian motion command.

Que: Queue the motion command for execution.

Mv: Start the motion task.

CSP: Calculate the next Set Point.

t8: Continue the motion task.

Sta: Stop the motion task.

Fin: Report the completion of the motion task or an error state.

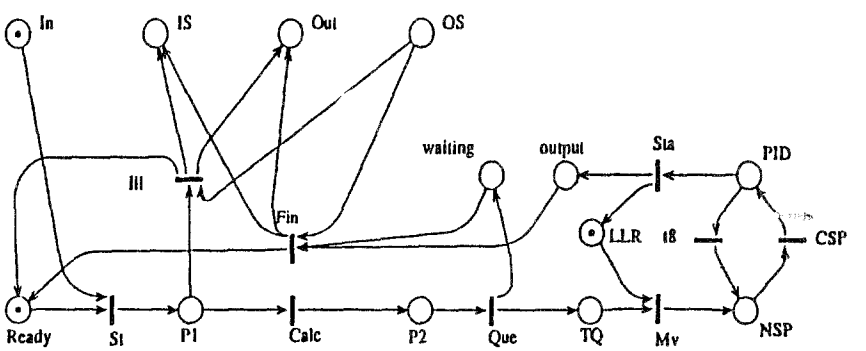

Fig. 5. The Petri net transducer for motion coordinator.

Places:

In, Out: The input and output places.

IS, OS: The input and output semaphores.

Ready: Motion system ready for the motion command.

PID: Executing PID control.

Waiting: Waiting until the lower levels of Kali output either the desired position or an error.

Output: The lower levels of Kali have either reached an error or some position.

LLR: Ready for the next job in the motion queue.

P1: Verifying the legality of path.

P2: Ready to enqueue a motion task.

TQ: Task Queued.

NSP: Get the Next Set Poit.

The set of subtasks to be processed by the motion coordinator is Σ_m. The translation mapping σ_m for the motion coordinator can be specified as,

Translation $\sigma_m(Mv, move)$:

```
set_Kali_mode(CARTESIAN);
load_from_queue(desired_path).
```

Translation $\sigma_m(Mv, approach)$:

```
set_Kali_mode(JOINT);
load_from_queue(desired_path).
```

4.4. PNT FOR THE GRIPPER COORDINATOR

There are 7 places and 7 transitions in the Petri net model of the gripper coordinator (Figure 6). The places and transitions are specified as follows:

Transitions:

Start: Start the grasping task.

Cross: Read the crossfire sensor, returning true or false.

GoP: Close the gripper to a desired width.

MeP: Measure the size of the gripper opening.

GoF: Close the gripper until the desired force is reached.

MeF: Measure the force that the gripper is applying

Next: Continue if there are commands left on input tape.

Finish: Report the result of task execution.

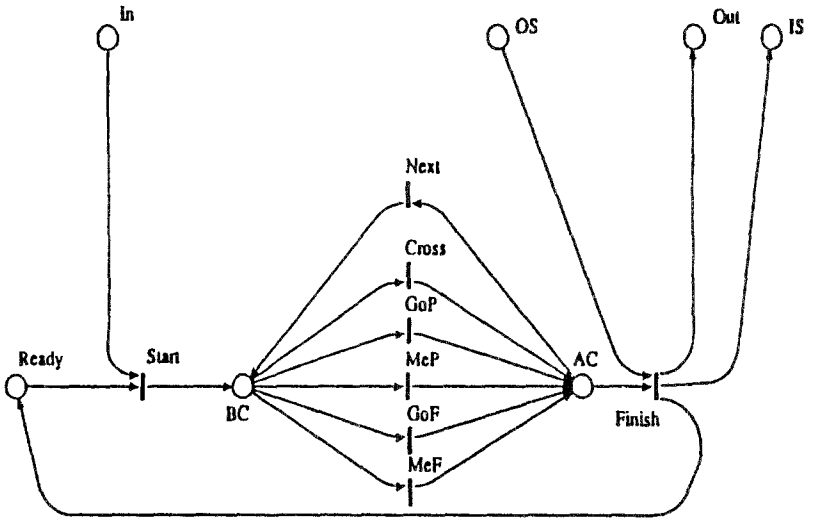

Fig. 6. The Petri net transducer for gripper coordinator.

Places:

In, Out: The input and output places.

IS, OS: The input and output semaphores.

Ready: Ready for grasping tasks.

BC: Starting task execution.

AC: Finishing task execution.

The set of subtasks to be processed by the gripper coordinator is Σ_g. The translation mapping σ_g is specified as,

Translation $\sigma_g(Cross, crossfire)$:

```
crossfire(){
if (data_from_crossfire_sensors == BLOCKED) return (TRUE);
else return(FALSE);
}
```

Translation $\sigma_g(GoP, goto\_position)$:

```
goto_position(desired_position){
      write_val = calculate_controller_value(desired_position);
      write_to_controller(write_val);
      if(whatever_the_error_conditions_are) return(ERROR);
      else return (OK);
}
```

Translation mappings $\sigma_g(GoF, goto\_force)$:

```
goto_force(desired_force){
      write_val = calculate_controller_value(desired_force);
      write_to_controller(write_val);
      if(whatever_the_error_conditions_are) return(ERROR);
      else return (OK);
}
```

Translation $\sigma_g(MeP,\ read\_position)$ and $\sigma_g(MeF,\ read\_force)$ are performed by the routine **measure_position()** and **measure_force()**, respectively.

5. Communication Mechanism and Simulation

A CAD tool called *TokenPasser* has been developed to allow the system designer to define a distributed hierarchy of communicating PNTs (Mittmann, 1991). *TokenPasser* allows us to distribute PNTs on different Unix machines with easy ways of changing the connections between different PNTs or the machine on which a PNT runs. This section first discusses the issues related with communication and then describes the results of simulation conducted by using *TokenPasser*.

5.1. COMMUNICATION MECHANISM

The communication between the processors running PNTs is an important issue. Since all CIRSSE machines are running under the Unix operating system and are connected by a Thinnet Ethernet, so most Unix methods of communication are easily available.

A socket is a Unix Inter Process Communication (IPC) construct, which allows communication between processes on multiple machines. Sockets allow many options including non-blocking reading and writing, and asynchronous notification of data arrival. Functionally sockets look like files, they can be written to, or read from with the **read**(2) or **write**(2) commands. Sockets have many communication semantics, the most convenient one was the SOCK_STREAM which provides sequenced, reliable, two-way byte streams. Sockets also use several different protocols, the most relevant one is PF_INET, which is an internet protocol and provides enough versatility to go across Ethernets.

Since all UNIX IPC is built upon sockets, sockets are the fastest built-in communication method possible, and due to the relative ease of writing and reading data using them, SOCK_STREAM, PF_INET sockets are the communication tool used.

5.2. SIMULATION RESULTS

Simulations has been conducted with *TokenPasser* to test the speed of communication when the size of message and distances between the machines are varied. The default setup of simulation is:

Dispatcher: Sol, a Sun 4/260, which is the system file server and network host.
Vision Coordinator: Venus, a Sun 4/60 Sparc Station.
Motion Coordinator: Mars, a Sun 3/260.
Gripper Coordinator: Earth, a Sun 3/150.

Timing of the speed has been done by attaching a routine to every transition in the dispatcher, and recording the amount of time passed (in *millisecond*, ms) since that routine was last called. All of the times which measure the delay between sending a message and receiving one back have been recorded.

The task plan (or string) processed by the dispatcher in the simulation is the following: *cal_r*, *cal_v*, *move*, *find_obs*, *move*, *approach*, *move*, *find_sn*, *move*, *approach*, *release*, *approach*, *grasp*, *move*, *approach*, *release*.
This has resulted in 52 transitions being fired, causing 26 message transmission delays.

5.2.1. Message Size Variation

Variation in transmission delay when the message size is increased has been tested. Under normal operations, sending a token involves the transmission of 8 bytes, and a sequence of commands 12 to 40 bytes. In this test 50 and 500 bytes were appended to each socket message to see if this deteriorated the speed of the message transmission.

The results of the test (Figures 7a–7c) have indicated that adding 50 bytes didn't affect the performance, while adding 500 bytes doubled the average delay time (from about 250 ms to about 500 ms).

5.2.2. Network Configuration Variation

Variation in transmission delay due to change in the distances between the computers which are running the PNTs and changes in how the computers are connected has also been tested.

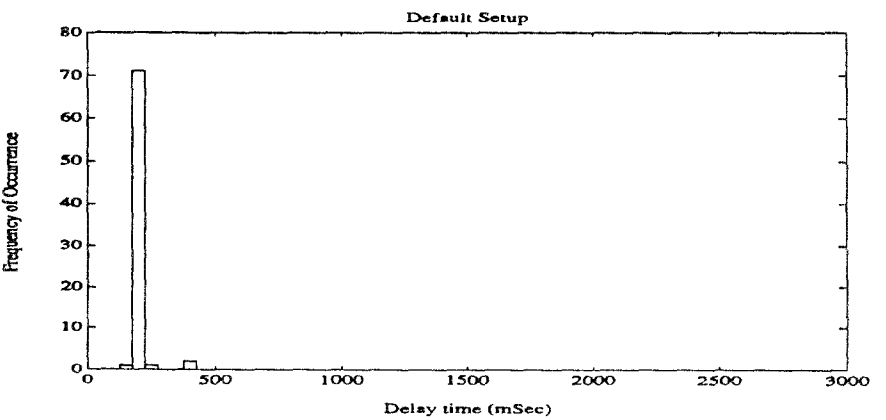

Fig. 7a. Distribution of delay time with message size variation for default setup.

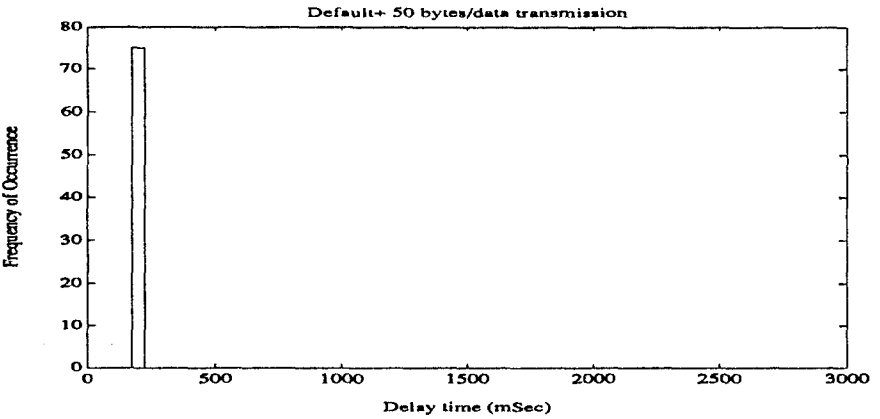

Fig. 7b. Distribution of delay time with message size variation for default setup.

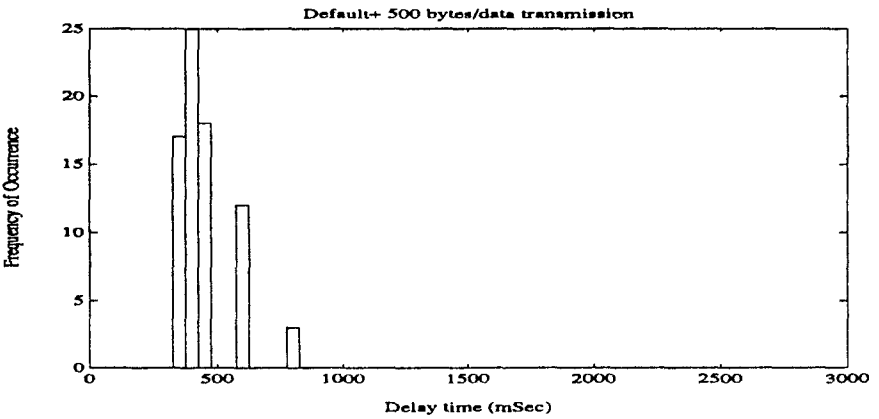

Fig. 7c. Distribution of delay time with message size variation for default setup.

All PNTs Local. In this test all of the nets are running on Sol, communicating, as normal, via sockets. The results of the test can be seen in Figure 8. Clearly, by comparing with the result of the default setup, there is an increase in delay in this case.

The explanation for the counter-intuitive increase in delay can be seen when one realizes that a computer only checks sockets every 200 ms, and since a computer will be synchronized with itself, (and is unlikely to be synchronized with another computer), a round trip socket communication on one computer takes twice as many clock cycles as communication between different computers.

On Peers. In this test the dispatcher is moved to Moon (a Sun 4/60GX Sparc Station) to see if the dispatcher being the net server improved or deteriorated the situation.

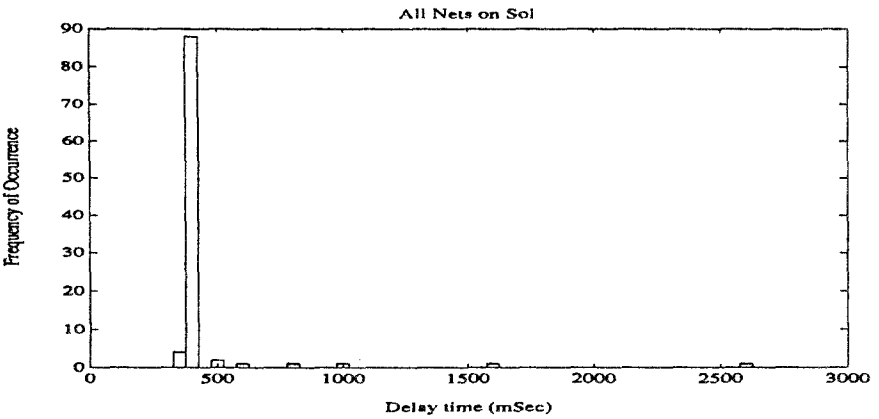

Fig. 8. Distribution of delay time when all nets on a single machine.

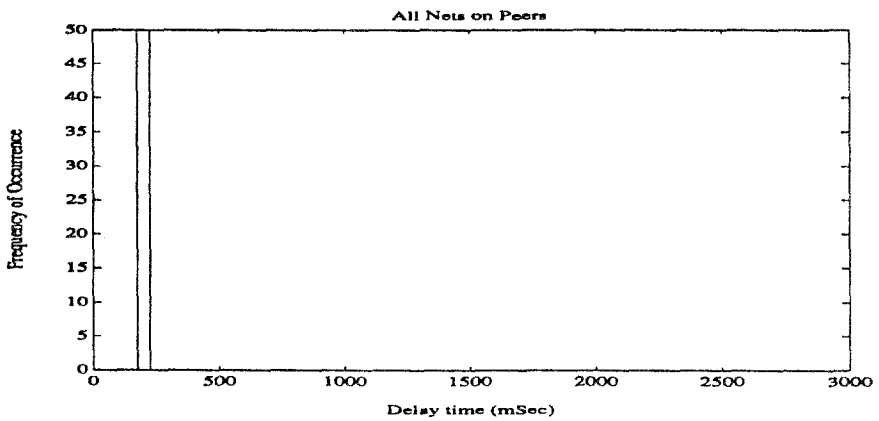

Fig. 9. Distribution of delay time when dispatcher on the file server.

As one can see from Figure 9 and Figure 7a, there is no significant difference in transmision delays, thus relieving any necessity of placing the dispatcher on the (possibly overloaded) file server.

Nonlocal Dispatcher. The dispatcher was placed on *pawl3.pawl.rpi.edu,* a Sun 3/50, located across campus. The message size was increased by 50, then to 500 bytes in the test.

As one can be seen from Figures 10a–10c, the increased distance produced almost no changes in the normal and +50 byte tests, however when the amount of data transmitted gets larger, the increased distance produces a more pronounced effect. In fact, when the program was run with the display turned on (necessitating increased data transmission to allow the graphics) the program halted with lost data on 3 of

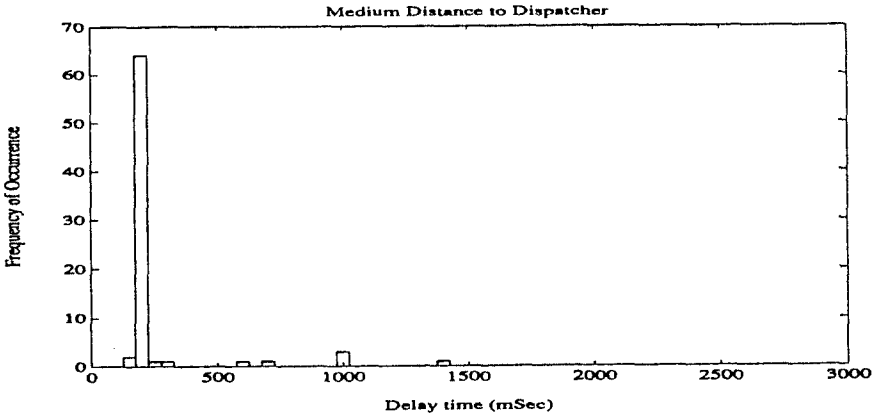

Fig. 10a. Distribution of delay time with message size variation for nonlocal dispatcher.

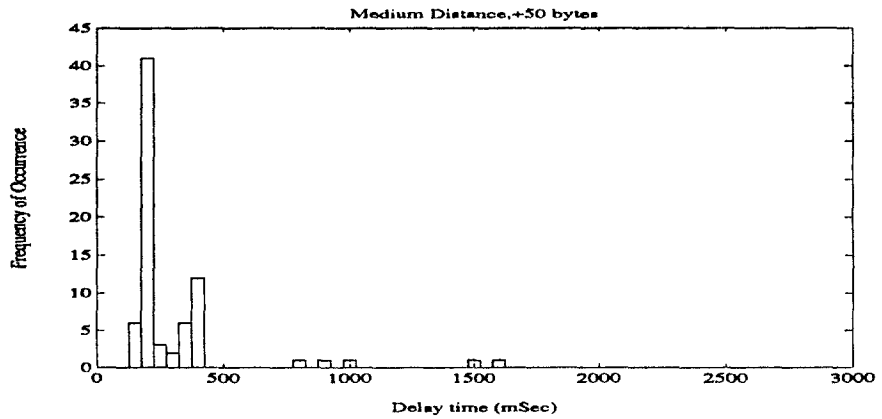

Fig. 10b. Distribution of delay time with message size variation for nonlocal dispatcher.

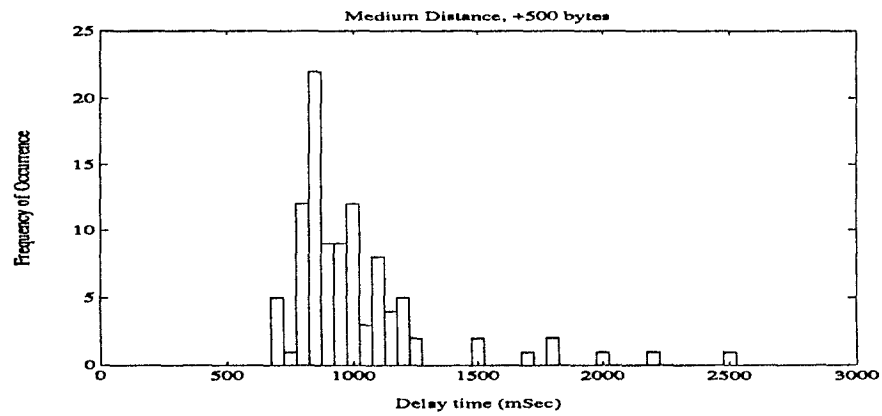

Fig. 10c. Distribution of delay time with message size variation for nonlocal dispatcher.

the 4 attempts. In further testing it was observed that more displays resulted in less time to failure.

Remote Dispatcher. The coordinators were located on their normal computers, resulting in about 3000 miles of Internet communication between the PNTs.

From Figures 11a–11c, it can be seen that this increased distance resulted in degraded communication speed even with minimal data. The PNTs still ran (although more slowly) with 50 extra bytes of data, even when displaying full graphics. However the increase to 500 extra bytes of data per socket transmission resulted in the PNT failing to complete the task every time (in 6 consecutive trys). When the extra load was lowered to 200 bytes, it resulted in success in 5 out of the 6 attempts.

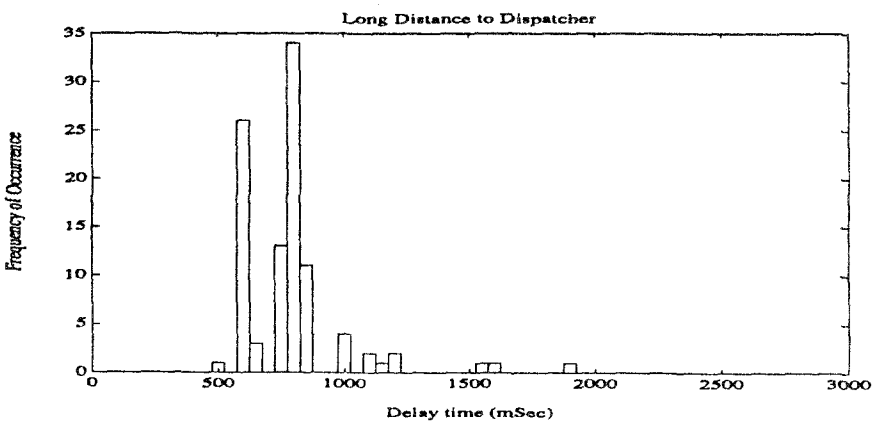

Fig. 11a. Distribution of delay time with message size variation for remote dispatcher.

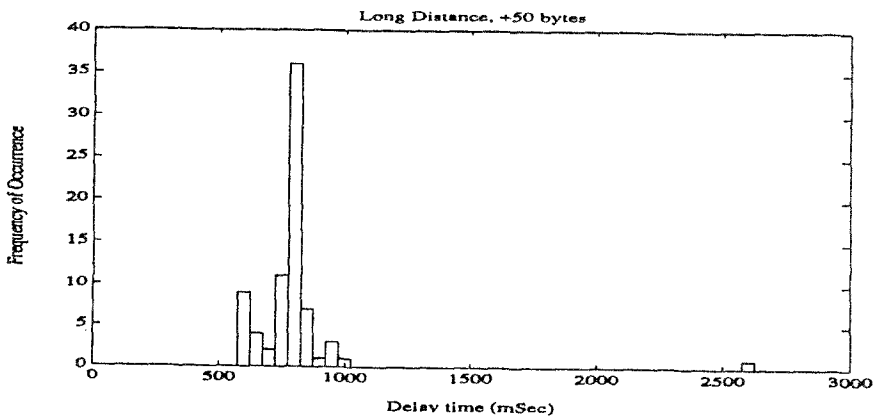

Fig. 11b. Distribution of delay time with message size variation for remote dispatcher.

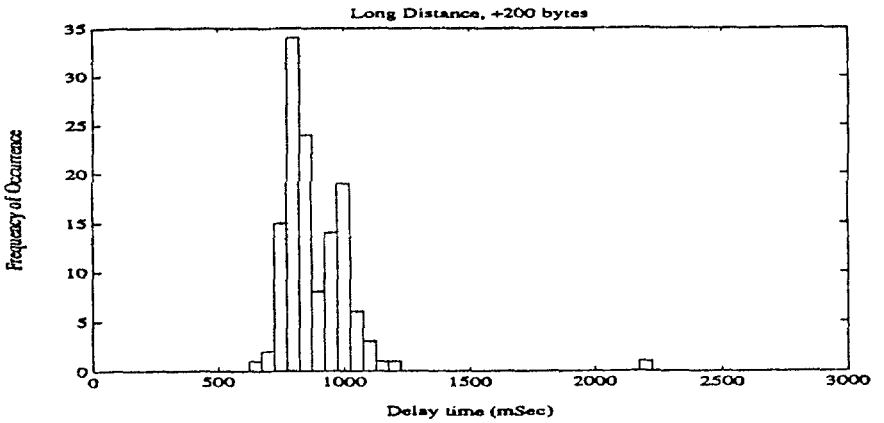

Fig. 11c. Distribution of delay time with message size variation for remote dispatcher.

6. Conclusions

This paper has presented a formal specification of the coordination level of the CIRSSE Robotic Platform System by using Petri Net Transducers to specify a coordination structure in which one dispatcher supervising three coordinators. A simple task grammar for the command language has been described and used in the construction of the PNT model of the dispatcher. All four PNTs for the dispatcher and coordinators have been defined in detail and their task translations have been illustrated by pseudo *C* codes. The simulation conducted on the communication speed within the Coordination Level has indicated that the longer the physical distance between the computers which host the dispatcher and coordinators, the smaller the number of tokens (or amount of information being transferred) required to degenerate the communication speed. This observation agrees with our intuition and once again demonstrates the importance of developing an intelligent autonomous control system for the CIRSSE Robotic Platform System, since such a control system will reduce dramatically the intensity of information exchange between the earth station and the space station.

Acknowledgement

This work was supported by the NASA Center for Intelligent Robotic Systems for Space Exploration (CIRSSE) under Grant # NAGW-1333. The first author has been partially supported by a Special Purpose Grant in Science and Engineering Program from AT&T Foundation.

References

Azema, P., *et al.* (1984), Specification and verification of distributed systems using prolog interpreted Petri nets, *Proc. 7th Int. Conf. Software Eng.*, Orlando, U.S.A, pp. 510–518.

Boehm, B.W. (1976), Software engineering, *IEEE Trans. Computers* **C-25**(12), 1226–1241.

Bruno G. and Marchetto, G. (1986), Process-translatable Petri nets for the rapid prototyping of process control systems, *IEEE Trans. Software Eng.* **SE-12**(2), 346–357.

Courvoisier, M., Vallette, R., Bigou, J.M., and Esteban, P. (1983), A programmable logic controller based on a high level specification tool, *Proc. 1983 Conf. Ind. Electron.*, pp. 174–179.

Krogh, B.H., Wilson, R., and Pathak, D. (1988), Automated generation and evaluation of control programs for discrete manufacturing processes, *Proc. IEEE Int. CIM Conference*, Troy, NY, U.S.A., pp. 92–99.

Mittmann, M. (1991), Token Passer: A Petri net specification tool, MS Thesis, ECSE Dept, RPI, Troy, NY.

Noreils, E.R. and Chatila R.G. (1989), A general structure for mobile robot action control, *IEEE/RSJ Int. Workshop on Intelligent Robots and Systems*, Sept. 4–6, 1989, Tsukuba, Japan, pp. 550–556.

Saridis, G.N. (1977), *Self-Organizing Stochastic Control Systems*, Marcel Dekker, New York.

Saridis, G.N. and Stephanou, H.E. (1977), A hierarchical approach to the control of a prosthetic arm, *IEEE Trans. Systems Man Cybernet.* **SMC-7**(6), 407–420.

Saridis, G.N., Foundation of intelligent controls, *Proc. IEEE Workshop on Intelligent Contr.*, RPI, Troy, NY, pp. 23–27.

Saridis, G.N. and Valavanis, K.P. (1988), Analytic design of intelligent machines, *Automatica* **24**, 123–133.

Topper, A., Caneshmend, L., and Hayward, V. (1988), A computing architecture for a multiple robot controller for space applications Kali Project. *Fifth CASI Conference on Astronautics*, Ottawa.

Valavanis, K.P. and Saridis, G.N. (1991), Probabilistic modeling of intelligent robotic systems, *IEEE Trans. Robot. Automat.* **RA-7**(1), 164–170.

Wang, Fei-Yue and Saridis, G.N. (1990), A coordination theory for intelligent machines, *IFAC J. Automat.* **26**(9), 833–844.

Wang, Fei-Yue and Saridis, G.N. (1991), Petri net transducers for task translation in intelligent machines, *Proc. IFAC Int. Workshop on Discrete Event Systems Theory and Applications in Manufacturing and Social Phenomena (DES'91)*, June 1991, Shenyang, China.

Wang, Fei-Yue and Saridis, G.N. (1992), *Coordination Theory for Intelligent Machines: Applications in Intelligent Robotic Systems and CIM Systems*, to be published by Kluwer Academic Publishers, Boston, MA.

Wang, Fei-Yue and Saridis, G.N. (1993), Task translation and integration specification in intelligent machines, *IEEE Trans. Robot. Automat.* **9**(3), 257–271.

Wang, Fei-Yue (1990), *A coordination theory for intelligent machines*, PhD Thesis, ECSE Dept, RPI, Troy, NY.

Wang, Fei-Yue, Kyriakopoulos, K.J., Tsolkas, A., and Saridis, G.N. (1991), A Petri net coordination model for an intelligent mobile robot, *IEEE Trans. Systems Man Cybernet.* **SMC-21**(4), 777–789.