NASA-CR-191862

IN-65-CR

145738

P.146

A PETRI NET CONTROLLER FOR DISTRIBUTED HIERARCHICAL SYSTEMS

NAGW-1333

CIRSSE

Center for Intelligent Robotic Systems for Space Exploration

Rensselaer Polytechnic Institute
Troy, New York 12180-3590

A PETRI NET CONTROLLER FOR DISTRIBUTED HIERARCHICAL SYSTEMS

NAGW-1333

by

Joseph E. Peck

Rensselaer Polytechnic Institute
Electrical, Computer, and Systems Engineering Department
Troy, New York 12180-3590

December 1991

CIRSSE REPORT #109

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENT

I would like to thank Professor Alan Desrochers for the opportunity he gave me to work for CIRSSE, as well as for his support and guidance during my research. I would also like to thank Don Lefebvre for his assistance in developing the Petri net requirements and in adapting the software to the CTOS environment. My appreciation goes out to Keith Fieldhouse for his assistance in tracking down some subtle programming errors. I also wish to thank Steve Murphy for his last minute assistance in generating the testbed figures. Finally, I need to thank Mary Sargent for her continual support throughout my academic career.

CHAPTER 1
Introduction

1.1 Motivation

The contents of this paper and the associated software came about as a direct result of certain needs for the Center for Intelligent Robotic Systems for Space Exploration (CIRSSE) dual arm robotic testbed at Rensselaer. Relatively independent research groups have been working on various subsystems of a robotic platform capable of supervised autonomy. Each of these subsystems have matured to the point that significant effort is now going into the integration of the components for high level testing and development. To successfully integrate the testbed, tools were required to 1) model the system; 2) integrate the subsystems; and 3) obtain user interaction. This paper presents a Petri net based controller which satisfies these three needs. An underlying theme in each of these requirements is flexibility, such that future expansion and development of the system will not render today's controller obsolete.

1.2 The CIRSSE Dual Arm Robotic Testbed

The architecture of the CIRSSE dual arm robotic testbed was developed to provide a platform for research and development of a space-based robotic system [9] [26]. The hardware consists of two PUMA manipulators connected to carts on a common linear track, with a laser and five camera vision system. The manipulator hardware is shown in Figure 1.1. Software servicers have been developed to perform the low level interactions with the hardware, providing a level of abstraction for simplified application development. This testbed is currently supporting a number of research projects, including: visual object recognition; visual servoing; strut

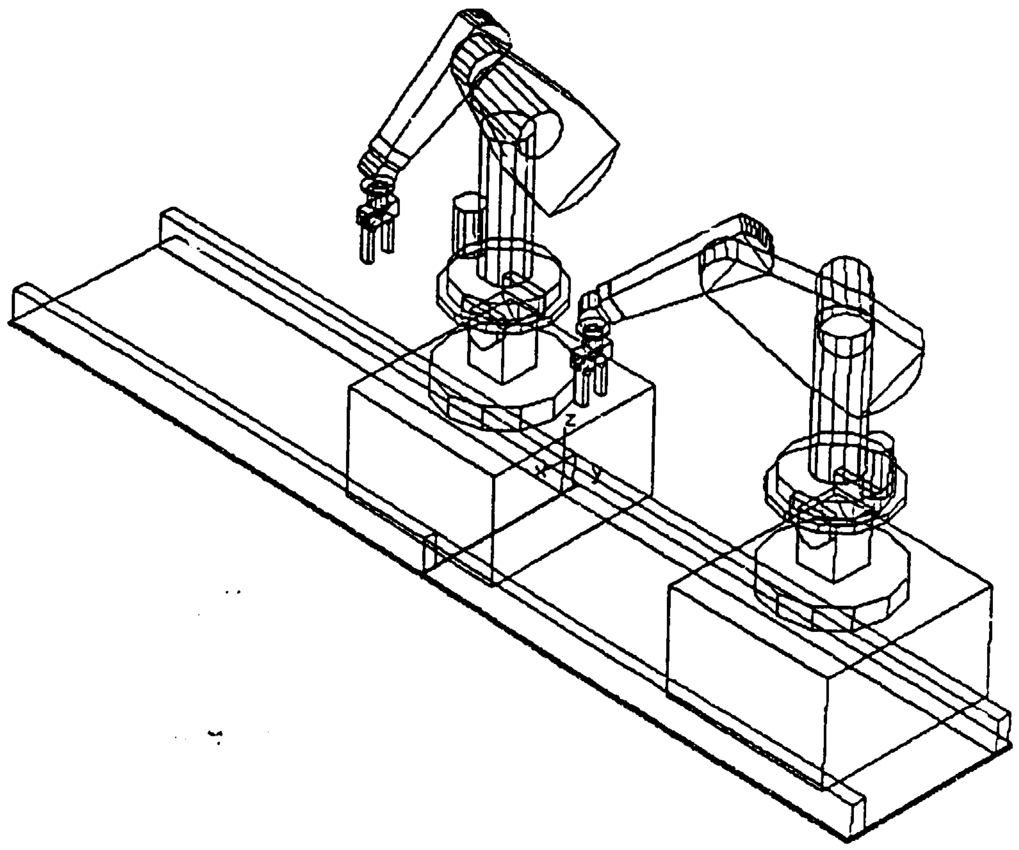


Figure 1.1: The CIRSSE Dual Arm Robotic Testbed

and node assembly and disassembly; two arm coordination and control; and path planning with collision avoidance.

1.3 Thesis Organization

Chapter 1 presents the motivation behind this thesis, and briefly discusses the CIRSSE testbed. Chapter 2 provides a brief description of current Petri net theory, tools and controllers. Chapter 3 presents the problem description, as well as the proposed implementation of the solution. Chapter 4 details the use of the controller itself, both in setting up and actually running the software. Chapter 5 discusses the controller's conformity to Petri net theory. Chapter 6 presents two case studies in the CIRSSE testbed environment. Chapter 7 summarizes the results of this project, and provides directions for future work. Appendix A contains the descriptions of the currently available enabling and post functions. The software for this controller implementation is contained in the remaining appendices. Appendix B contains the header files, Appendix C contains the source code for the player task, and Appendix D contains the source code for the display task.

CHAPTER 2
CURRENT PETRI NET THEORY AND APPLICATIONS

2.1 Introduction

This chapter describes several aspects of Petri net theory relevant to this project. Current tools aiding the design and analysis of Petri nets are discussed, and two existing Petri net controllers are presented.

2.2 Petri Net Theory

Petri nets are a graphical and mathematical modeling tool for discrete event dynamic systems [12] [14]. They are particularly well suited to modeling asynchronous, synchronous, and concurrent actions, which occur extensively in most large robotics and automation applications [1] [15]. One representation of a Petri net is a weighted bipartite directed multigraph, whose node types are called transitions and places. Places and transitions are interconnected by arcs, termed input arcs and output arcs with respect to the connected transition. Each place may contain some non-negative quantity of tokens, and this marking indicates the state of the system. The firing of a transition represents an event that changes the state of the system, thereby changing the marking of the places. A Generalized Stochastic Petri Net (GSPN) is an ordinary Petri net whose transitions may take a deterministic period of time to fire. A transition is enabled to fire when each of its input places contain at least as many tokens as the weight of its associated input arc. An enabled transition may fire at any time, and that firing consists of three phases: 1) the enabling tokens are removed from the input places; 2) some action may be performed which may cause a time delay; and 3) once the action is complete, tokens are added to the output places.

A Petri Net Transducer (PNT) [22] [24] is an extension of a GSPN which adds another enabling condition to the firing requirements of a transition. An input tape composed of members of the input alphabet is passed to the PNT, whose transitions possess an optional tape requirement. A transition with a tape requirement will be enabled only if the head of the tape contains the proper command and its token requirements are met. These extensions make PNT's a more powerful modeling tool, but it does make the analysis of the Petri net more complicated.

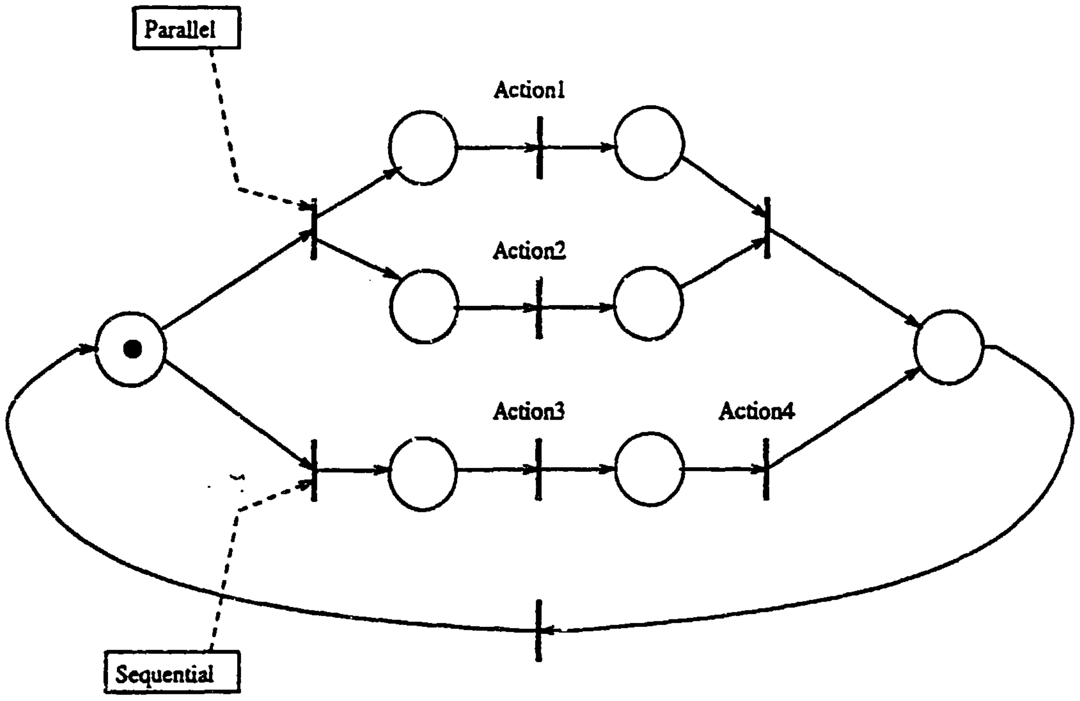


Figure 2.1: An Example Petri Net Transducer

An example PNT is shown in Figure 2.1. The two initial transitions each require a different tape command in order to fire. If Parallel is at the head of the tape, the upper path will be taken, while if Sequential is at the head the lower path will be followed. Both paths perform two actions, but the upper path performs Action1 and Action2 concurrently, while the lower path performs Acti~~^n~~3 and

Action4 sequentially.

2.3 Petri Net Tools

Although Petri nets have been used extensively in manufacturing environments, to date there is no one package that provides full design and analysis capabilities. Petri nets are inherently graphical, and lend themselves nicely to Computer Aided Design entry. However, the analysis capability of the existing tools seems to be inversely proportional to their ease of use and quality of interface.

2.3.1 GreatSPN

GreatSPN [3] offers a graphical user interface for the construction, editing and analysis of Petri nets. This package was developed at the University of Torino, Italy, as a tool for the specification and performance analysis of computer systems, and has since evolved into a general Petri net tool. GreatSPN supports immediate, stochastic and timed transitions, as well as inhibitor arcs. The Petri net can be analyzed for basic properties, such as p and t invariants, boundedness, steady state token distribution, and deadlock. Performance can be determined as well in terms of token throughput. In addition to the screen display, the software can also save the net in a postscript file for printing hardcopy output.

Note that GreatSPN is not without drawbacks. Its analysis capabilities are not fully implemented yet, and the abilities of the existing analysis options degrade with larger Petri nets. Finally, although the graphical user interface represents an advance over typical text based approaches, some aspects of the interface are still awkward to use.

2.3.2 SPNP

SPNP [4] is a text based Petri net analysis package developed at Duke University that provides a wide variety of analysis capabilities. The Petri net is described as a series of C function calls, which also define the type of analysis desired. SPNP builds the reachability graph, and from this it can determine the steady-state Markov chain, token distributions and throughput, and a host of other parameters.

Although the SPNP software itself can analyze a vary large net, entering the C function calls is error prone and difficult to verify. One approach to this problem is to use a graphically oriented design program, and convert the stored information into the desired format. GreatSPN2SPNP [13], written at Rensselaer Polytechnic Institute, is a conversion program that translates a GreatSPN output file into a suitable SPNP input file.

2.3.3 Xpn

Xpn [8] is a Petri net design tool being developed at Rensselaer Polytechnic Institute featuring a full graphical user interface running under X-windows. The created Petri nets may be stored in a variety of formats, suitable for use with other packages such as GreatSPN, SPNP, as well as the controller presented in this thesis. Note, however, that Xpn offers no analysis capability and is solely used for the net design. Currently, this package is under development, and does not yet fully support all file formats.

2.4 Petri Net Controllers

Petri nets have been used in industry for the modeling of systems, and many dedicated controllers have been developed with the aid of those models. Recent work at CIRSSE has been geared towards developing general Petri net controllers, for use in distributed hierarchical systems. Two such controllers developed at CIRSSE were submitted as Masters Projects in 1991. Although both controllers were designed to solve the same general problem, they possess very different implementations. These two, along with a manufacturing implementation, are described below to highlight their differences and to provide a basis of evaluation for the controller presented in this thesis.

2.4.1 Petri Net Controller

The Petri Net Controller (PNC) [5] [16] was developed to ease the difficulty in organizing and controlling manufacturing stations. It was one of the first implementations to abstract itself from a direct hardware interface, and instead executed on a general purpose computer. The capabilities of PNC are based within the places of the Petri net, where different types of places are used to represent and perform different actions. Macro places represent an embedded subnet, action places interact with the external world, and several other types exist as well.

Although a hierarchical arrangement of Petri nets is possible through the use of macro places, this organization is still executed on only a single computer. There are no built in means of distributing the controller itself over multiple machines. Additionally, the macro places allow for only one entry point to a particular subnet, which limits flexibility. Finally, the execution algorithm as implemented does not preserve the Petri net property of fairness, and special care must be made in designing each Petri net.

2.4.2 TokenPasser

TokenPasser [11] was designed to allow for the rapid prototyping of a distributed hierarchical control system which utilizes Colored Petri Net Transducers. Individual Petri nets, each with their own display, can be distributed on different computers. Each transition can execute a function call whenever it is fired, allowing for basic controller operation. The implementation of TokenPasser does not lend itself to use as a full fledged controller. However, it does provide performance data through simulation, which can be extremely useful in determining communication delays between various configurations. Designing a new Petri net application requires that the user write a series of C function calls to be compiled and linked with the controller software which define and initialize the net structure.

2.4.3 DPNC

Distributed Petri Net Controller (DPNC) [2] is a fuller controller implementation for distributed systems than TokenPasser, designed with control as a main objective, as opposed to simulation. A series of players are connected in a token ring fashion, with one central display. This controller executes a single Petri net, with individual sections being executed on the separate players. DPNC uses GreatSPN to design the Petri net, and the resulting definition file is converted into C functions by a UNIX script. These C functions are compiled and linked with the controller code itself, as is the case with TokenPasser. Additionally, any function calls that are associated with the firing of a transition must be written, compiled and linked as well. The single display is both a blessing and a curse. The user need only concern himself with one window on the screen, and all action occurs in this one location. However, the Petri net for any real controller will be far too large to fit on a single screen, and any attempt to do so will result in a cluttered and confusing representation. Like PNC, this controller does not preserve the Petri net property of fairness, and requires careful Petri net design.

2.5 Summary

Petri nets are a powerful tool for the modeling and analysis of discrete event dynamic systems. A Petri net based controller, with a fixed embedded Petri net or some more flexible arrangement capable of using an arbitrary Petri net, can be designed, analyzed and verified before the actual implementation. Additionally, any desired modifications to the controller can be evaluated before implementation as well. Tools such as GreatSPN may be used to graphically design and modify the Petri nets, while other tools such as SPNP can be used for detailed analysis.

To date, only PNC has been used in conjunction with physical hardware. It was used to control a machining workstation based on a Cincinatti Milacron 5VC Machining Center, which produced a variety of parts. The other controllers, TokenPasser and DPNC, were not developed sufficiently to be used in a real system. However, with additional work and effort they would have resulted in a more flexible means of controlling and monitoring a distributed, hierarchical system.

CHAPTER 3
PROBLEM STATEMENT AND SOLUTION

3.1 Introduction

This chapter details the requirements for the Petri Net controller, and discusses the chosen implementation. This controller supports a distributed, heterogeneous, hierarchical system with flexible execution and display options.

3.2 Project Specification

The requirements for this project are geared towards a general Petri net controller, and are not specific to its initial application, i.e., the CIRSSE testbed. First of all, it must support a distributed system. It is assumed that the application can execute on multiple processors, not necessarily in the same machine or even location. Note that the term distributed not only applies to the hardware in the system, but to the actual controller components as well. The overall system may be heterogeneous, in the sense that the individual computers may be running different operating systems or even vary in architecture. The controller must allow for hierarchical interconnections between the individual Petri nets, as a large class of systems are ideally controlled with this configuration. Graphical displays must be able to represent the actual executing Petri net, as well as subsets of places and transitions to provide a summary of the entire application. Finally, the development of a new application should not require the recompilation of the controller itself, i.e., its configuration should be data driven.

3.3 Project Design

The controller is broken into two distinct classes of processes: players and displays. The players are responsible for executing the Petri nets which actually perform the control of the system. As tokens flow through the net, the player performs any necessary interactions with other players and the external world. The displays are responsible for presenting the state of the system to the user, and provide some means of controlling the players. It is extremely important to recognize that the players and displays are completely separate entities, and that the players can execute even in the complete absence of the display processes. However, most applications will include the displays as a means of monitoring and controlling the execution of the players. To this end, the display contents are highly configurable to aid in the meaningful representation of the underlying player Petri nets.

3.3.1 Petri Net Transducer Extensions

The concept of PNTs is extended for this application, providing for a more efficient and powerful means of actually controlling a system. This extension is made in the form of an additional enabling requirement, called an enabling function. This enabling function returns a Boolean value indicating the state of an external condition. Although this extension can be modeled by adding an additional place plus inhibitor arcs to the Petri net for each possible tape command, it is not readily implemented as such. A second extension is the inclusion of post functions, which are executed with the firing of transitions. These post functions are the main mechanism of performing an external action. These extensions are shown in Figure 3.1, where the enabling condition of the transition consists of token requirements, an enabling function, and a tape command. The firing of the transition causes the post function to be executed.

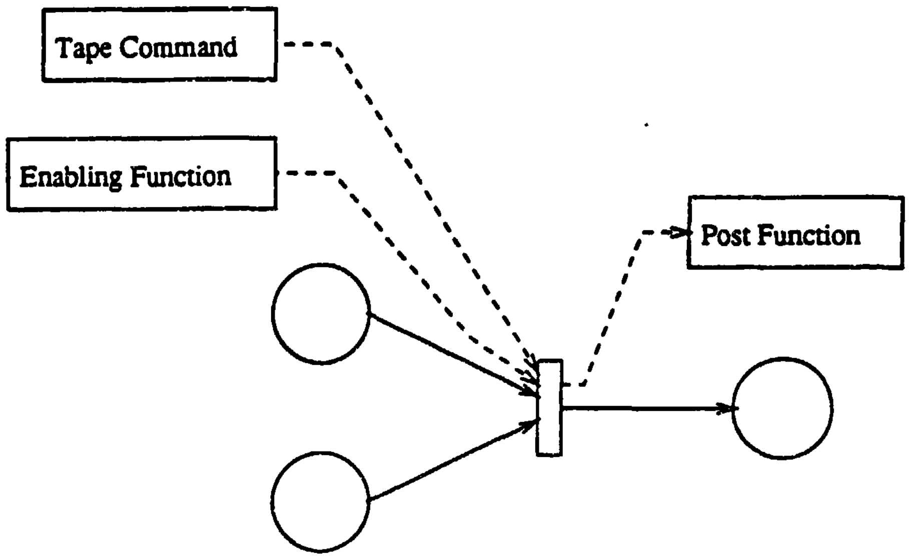


Figure 3.1: The Extended Petri Net Transducer

3.3.2 CIRSSE Testbed Operating System

The CIRSSE Testbed Operating System (CTOS) [6] [9] [26] was developed to facilitate the execution of applications which consist of multiple tasks and require intertask or interprocessor communication. CTOS greatly simplifies design of a distributed application, and forms the infrastructure for development on the CIRSSE testbed. CTOS is an extension of the base operating systems at CIRSSE, and has been implemented for both UNIX and VxWorks.

CTOS offers an environment for conveniently distributing tasks and executing a distributed application. A distributed application can be started with a single command line, through the use of configuration files. These application configuration files contain the list of tasks to be loaded and what machines to run them on, as well as any initial parameters to pass to the individual tasks. Reconfiguring an application involves simply editing this file, and rerunning the application. No other parameters need be changed.

Each task is assigned a symbolic name in the application configuration file, and it is this name that other tasks use to perform interprocess communication. If one task wishes to communicate with another, it first queries CTOS for the task *id* corresponding to the symbolic name of the other task. The task then uses CTOS message passing routines to perform the communication, where the task *id* is used to identify the recipient of the message.

Programs to be executed as CTOS tasks must be designed to exist in an event driven environment. These CTOS tasks are often called event handlers, due to their internal construction and external interface. In general, an event handler task is asleep until it receives a message indicating that there is some work for it to perform. The event handler is wakened, and performs the appropriate actions. It then puts itself back to sleep.

3.3.3 The Data Object Manager

One major concern of any distributed system is the location and accessibility of data objects. A Data Object Manager (DOM) [9] has been developed at CIRSSE to solve many data storage issues and provide a consistent interface to manipulating those objects. Any CTOS task can access the DOM, which offers a variety of capabilities. Data objects can be created, modified, renamed, deleted, copied and more. Oftentimes the existence of a particular data object is represented as a place in a Petri net; this place serves as a precondition for the firing of a transition which performs some action involving that object.

3.3.4 Player Design

The player task is responsible for the execution of the Petri net that actually controls the system. This involves the selection and firing of individual transitions, and interfacing with the tasks that perform actions. Another significant responsibility is notifying all interested display tasks of changes in the state of the Petri net. Figure 3.2 shows a high level view of the operation of the player code.

The player code spends the majority of its time in the Petri net execution phase. While there are no messages pending, the player is attempting to fire transitions. There are two execution modes, continuous and single-step. In continuous mode, the player repeatedly examines the enabling status of all transitions, and fires those that are enabled. In single-step mode, the player repeatedly examines the enabling status of those transitions which are not displayed in any display, and individually examines those transitions which are clicked on in the display. As soon as a message is received, the player exits from this enabling checking mode, and processes all of the pending messages. Once there are no more pending messages, it reenters the enabling checking mode. These messages include the arrival of tokens from other players, the toggling of player modes, and various CTOS commands such as application exit. The single-step mode is especially useful for debugging or user control of the system. Note that each player can be in either continuous or single-step regardless of the modes in the other players.

Petri net theory states that the firing of an enabled transition should occur without regard to the other transitions in the Petri net. There should be no particular order in which the transitions are examined for possible firing, or the execution of the Petri net may not follow the behavior predicted by the model. Therefore, in terms of theory an ideal implementation would randomly select which transition to examine as it executed the Petri net. However, a controller is also concerned with speed, and in a sizable net with a small number of enabled transitions, a pure random selection can result in an undesirable delay. A hybrid random-linear approach was taken. A random transition is selected to examine. If it is enabled, it is fired. Otherwise, the transitions are examined in order until an enabled transition

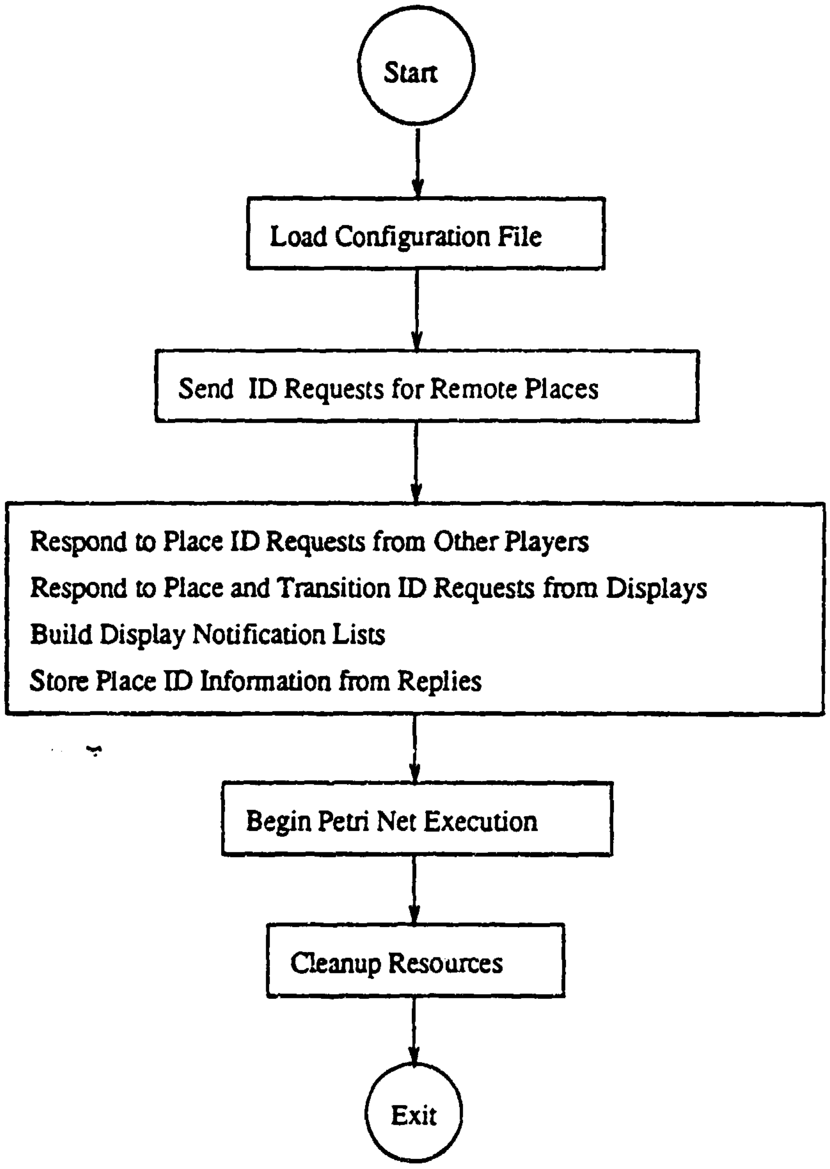

Start
Load Configuration File
Send ID Requests for Remote Places
Respond to Place ID Requests from Other Players
Respond to Place and Transition ID Requests from Displays
Build Display Notification Lists
Store Place ID Information from Replies
Begin Petri Net Execution
Cleanup Resources
Exit

Figure 3.2: Player Code Flow

is found and fired. After the transition is fired, the process begins again with the random selection. This provides for an arbitrary firing sequence, while maintaining real world performance.

3.3.5 Display Design

The display task provides the user with a graphical representation of the status of the system. These displays are highly configurable, and can serve a wide variety ot purposes. They do not need to mirror the underlying player Petri net, and can be used to provide a simplified representation of a particular net, or even a global representation of the entire collection of nets. Figure 3.3 shows a high level view of the operation of the display code. The parallel execution blocks in the flow indicate where a monitoring task is spawned. This spawned task sits in a loop monitoring the display window for user input and other X-window events. Whenever an event occurs that is of importance to the display event handler, a message is sent. The event handler then takes whatever action is appropriate. The display also receives messages from the various player tasks, indicating that some aspect of the Petri net status has changed. These can include tokens being removed or added to a place, or transitions beginning or ending to fire. Transitions in the display can correspond to two transitions in the player tasks, the firing of one causes the transition in the display to be highlighted, while the firing of the other completes some action and unhighlights the display transition.

3.3.6 Function Execution

One of the most important features of a controller is the ability to interact with the external world. Without this ability, the controller is reduced to a simulator. Due to this importance, an extremely flexible and powerful mechanism had to be developed to handle this interaction.

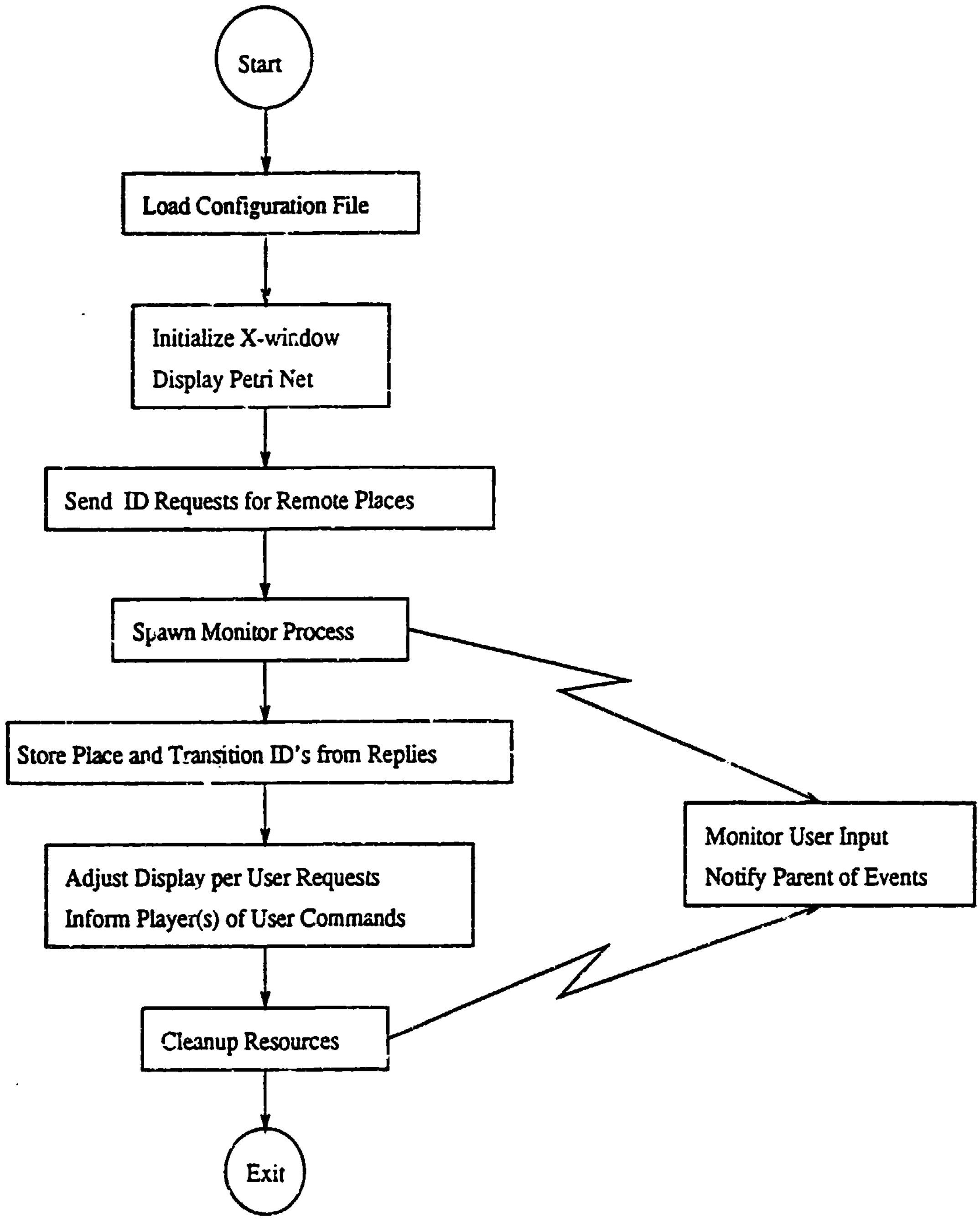


Figure 3.3: Display Code Flow

The controller supports enabling functions and post functions. Each of these is split into two classes, local or remote. The local functions are those that are compiled into the controller itself, while remote functions are those that are executed by a separate task. The local functions are geared towards looping or checking the status of some variable in the DOM. The external functions perform actions such as moving a robot, taking a stereo picture, etc. When a remote function needs to be executed, the player sends a message to the task capable of performing the operation, informing it of which particular action is requested and what data objects to use. The task name, action, and data objects are specified in the player configuration file.

3.3.7 Tape Issues

This controller fully supports the concept of a PNT input tape. Transitions can possess an optional tape command, which is an additional enabling requirement. These tape requirements can be used to parse a high-level language into lower level components, aiding in the decomposition of tasks. Although the player itself can accept an input tape and process its contents, no CTOS task exists for actually sending a tape to the player. As stated earlier, the tape commands can be implemented as a group of input places, and current Petri net designs in this thesis are utilizing that approach.

3.4 Summary

The controller has been designed to meet all of the current requirements of the CIRSSE dual arm robotic testbed, and in fact exceeds them in order to provide a viable system for future applications and research.

CHAPTER 4
USING THE CONTROLLER

4.1 Introduction

This chapter provides a brief user manual for designing and implementing an application utilizing this controller. Design concepts, as well as technical details concerning the related configuration files are presented and discussed. The commands available through the displays at runtime are also described.

4.2 Application Design Methodology

A typical application consists of three to five player tasks, and approximately the same number of display tasks. The primary work is performed in defining the player Petri nets and their interconnections. The display Petri nets are more often than not a stripped down version of the player Petri nets, with such structures as buffers or communication points omitted for clarity in the display.

The player Petri nets are interconnected via local and remote places. A local place is defined as a place that is native to the player Petri net in question, while a remote place is one which is native to some other player. This is best demonstrated with an example. Figure 4.1 shows three player Petri nets.

The first is a global net which performs two distinct actions. This net can be broken into two subnets, shown at the bottom. Note that the place **in_progress** appears in both subnets. The distinction is that **in_progress** is local to subnet 2, and remote to subnet 1. A remote place in a net does not exist as a regular place, but is actually a link to the local place in the remote net. The remote net can be on the same processor, a different processor within a single VME cage, or even across the ethernet network, and the link provides the interconnection to that Petri net.

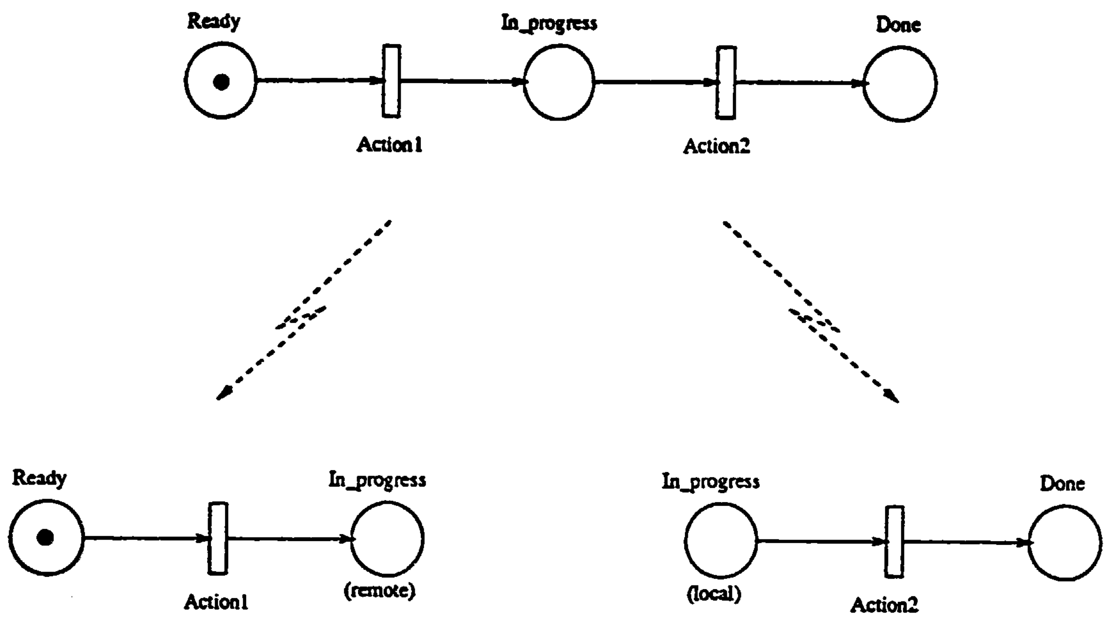


Figure 4.1: Petri Net Interconnection

The controller itself does not enforce the use of any particular interconnection scheme, however, one should be adopted if the analysis of the Petri net is desired. The two case studies presented at the end of this paper are organized in a hierarchical fashion, and a requirement for both is the ability to analyze the system and provide a consistent interface between modules. A consistent interconnection strategy provides two benefits: only one method of analysis needs to be established for the system as a whole; and the designers of new Petri nets in the system have a template for interconnection to follow. A hierarchical organization usually follows the principle of Increasing Precision with Decreasing Intelligence [17] [19]. The highest intelligence and global responsibility resides in the top of the tree, while the lowest level functional abilities lie along the bottom. Levels inbetween serve as coordinators, taking the higher level instructions and breaking them into a form usable by the lower level. This corresponds to a situation where we would like a single transition in a higher Petri net to request that a lower Petri net perform some action for it. The concept of this structure is shown in Figure 4.2, where the high level functions **Action1** and **Action2** correspond to the indicated subnets.

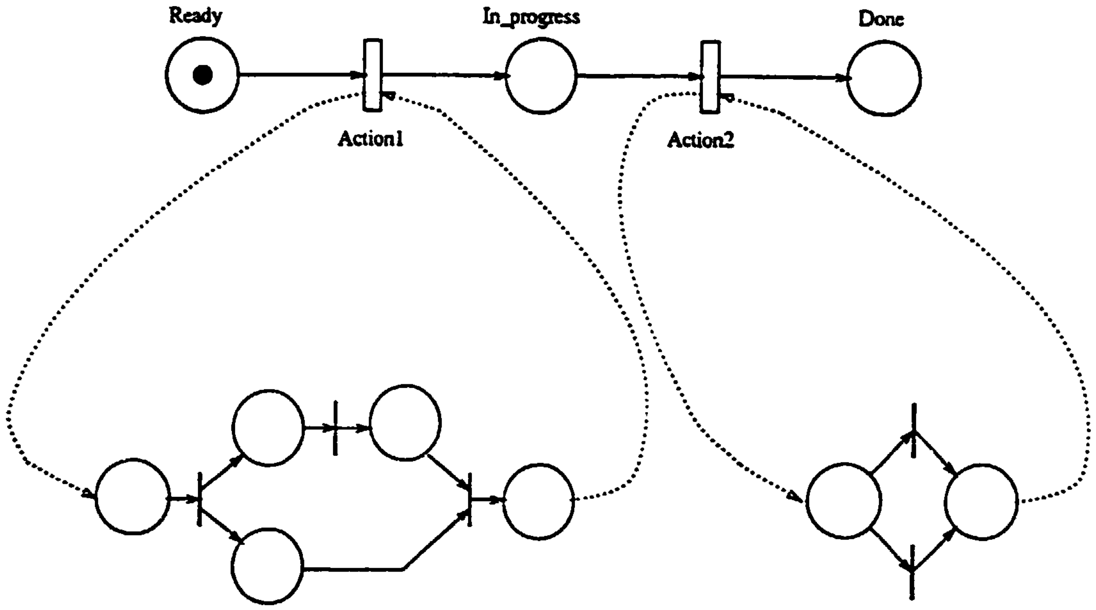


Figure 4.2: A Petri Net Hierarchy

Petri nets to be used in the CIRSSE testbed use a formal coordination structure to provide this functionality. Each level in the Petri net hierarchy can only perform a single action at a time for the level above it. This does not mean that parallelism cannot occur within each Petri net, only that a net can only satisfy one request at a time from the higher levels. This coordination structure is shown in Figure 4.3, detailing the necessary components in both Petri nets. The single transitions in Figure 4.2 have been replaced with two transitions and a place. The place is used to indicate that the requested operation is in progress. When the Petri net is available, the places avail and S have tokens in them. S is a semaphore indicating whether the net is currently performing some action. If not, **begin** can fire, placing a token in **in_progress** and I. It also places a token into one of the request action places, which indicates which operation is to be performed. This example Petri net can perform three possible actions, whose request arcs are shown as dotted lines. Only one of these arcs should actually exist for a given **begin** transition. The transition **start** can now fire, putting a token in **ready**. Now the first transition of the requested action may be fired. Once the operation is complete, a token is put in place **done**. The transition **finish** is then enabled, and upon firing puts a token in place O. This enables transition **end** to fire, which restores the semaphore and allows the Petri net to process another operation request.

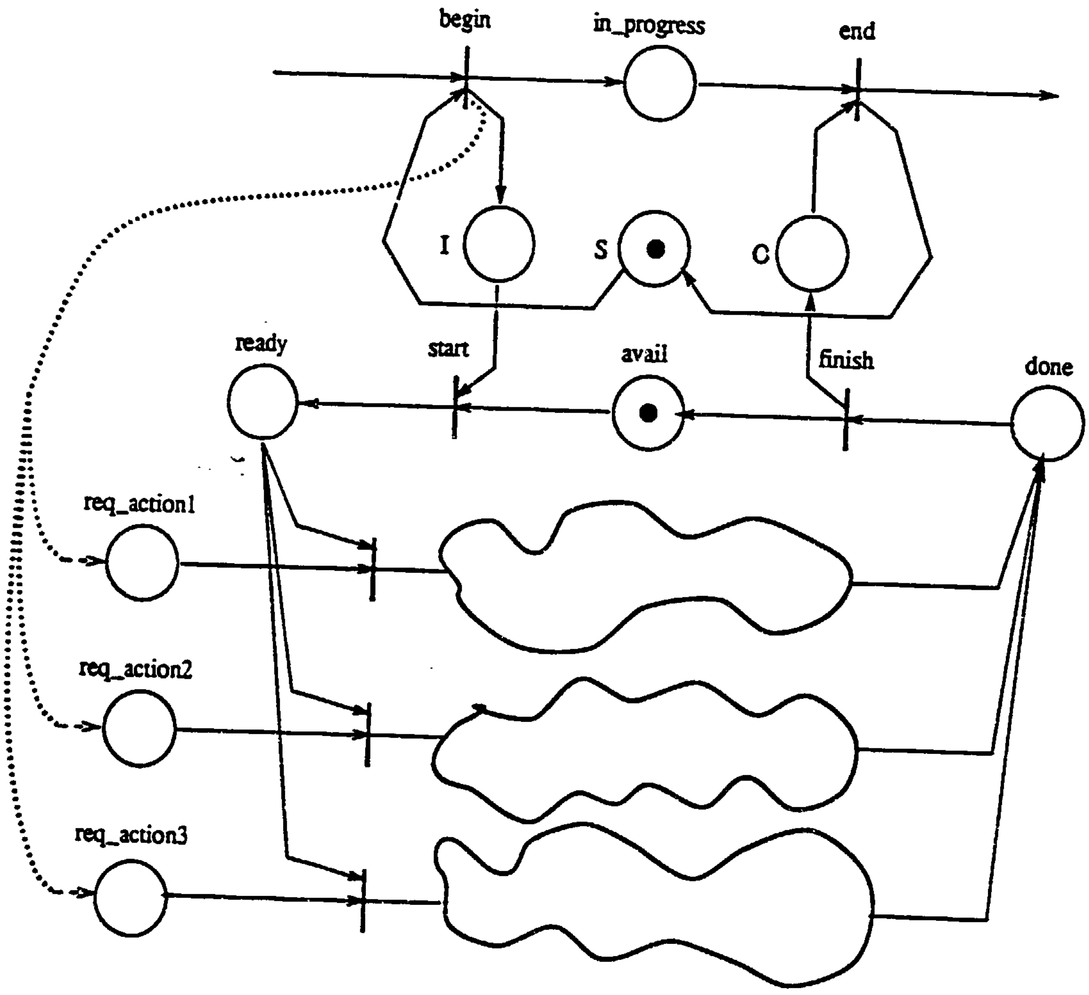


Figure 4.3: The Coordination Structure

4.3 Application Configuration File

The entire controller application consists of CTOS tasks, which are executed through the aid of a CTOS application configuration file. This file details the distribution of tasks, their symbolic names for interprocess communication, and any initial parameters required by the individual tasks.

The individual tasks may be executed on any machine in the network, so long as it is running CTOS. Prior to declaring which tasks to run, the machine to load them on must be defined. This is done with the line:

```
chassis machine_name
```

This indicates that the tasks after this line are to be started on the machine specified by machine_name. Multiple chassis lines may be placed in the file in order to distribute the tasks over the various machines.

The player task accepts only one parameter, which indicates which configuration file to use. Here are the lines corresponding to a player task and its parameter:

```
task  play_eh  player1
args  player1  net1
```

The keyword **task** indicates that this line defines a symbolic name for a task to be executed. The CTOS task play_eh is the player program, and in this example it is referenced by the symbolic name player1. The keyword **args** indicates that the line defines a parameter to be passed to a task. In this instance, the string **net1** is passed to the task with the symbolic name player1.

The display task can accept two parameters, one indicating which configuration file to use and one indicating which monitor to display on. Here are the lines:

```
task  disp_eh   display1
args  display1  net1  xterm6:0.0
```

Here the CTOS task **disp_eh** is the display program, and it is referenced by the symbolic name **display1**. The two parameters are passed using the same **args** line. The first parameter is the configuration file to use, and it is required. The second line is the actual display device to use. If the display device is omitted, the window will appear on the user's default device.

4.4 Petri Net Configuration Files

The Petri net configuration file is the mechanism used to provide the player and display tasks with the configuration and task interconnections of the Petri net that they are to execute or display. The configuration files differ in content between the player and display tasks, but their format is similar. In an effort to use existing tools for the design and analysis of a distributed system, it is recommended that the Petri nets be designed using GreatSPN, or another program such as Xpn that can generate the GreatSPN file format or this configuration file format directly. The configuration file begins with the GreatSPN **.net** file, and additional configuration data for each place and transition is appended to the end. This approach was taken to allow the user to modify this additional data with a standard editor until a customized design tool such as Xpn is fully developed and available. The file naming convention used by this controller is that all player configuration files end in **.pnet** and that all display configuration files end in **.disp**.

4.4.1 Player File

The player configuration file provides the details of the executable net. It is in this file that we specify the tape commands, enabling functions, and post functions for each transition. We also specify the owner task of every place referenced by a transition in this net.

The configuration data begins with the line:

```
SUBNET    task_name
```

This identifies the file as a player configuration file, and provides the symbolic name of the task for reference. Following this header, there is a line for every place and transition in the Petri net. The format for a place consists of:

```
PLACE    local_name    [LOCAL]|[REMOTE remote_name remote_owner]
```

A place with the LOCAL keyword is owned by this Petri net, and it is referenced by all other tasks as local_name. A place with the REMOTE keyword is owned by a different Petri net, and corresponds to the place named remote_name in the player with the symbolic name remote_owner. Here is the format for a transition:

```
TRANS    local_name    [TAPE command]
                       [[ENABLE_LOCAL function_name action data_name]
                       |[ENABLE_REMOTE task_name action data_objects]]
                       [[POST_LOCAL function_name action data_name]
                       |[POST_REMOTE task_name action data_objects]]
```

Transitions are always owned by the player that they appear in, and are not referenced by any other player tasks. Every transition may possess a tape command requirement, which is denoted by the keyword **TAPE** and followed by the name of the tape command. Every transition may possess either a local or a remote enabling function. Local enabling functions use the keyword **ENABLE_LOCAL**, and are followed by the local function name, an integer corresponding to the desired action, and a data name. A small data object manager exists in the player, which handles the local variables. Remote enabling functions use the keyword **ENABLE_REMOTE**, and are followed by the task name that performs the function, an integer indicating the desired action, and list of comma separated data objects.

Finally, every transition may possess either a local or a remote post function. The post functions use the same format as the enabling functions, but use the keywords **POST_LOCAL** and **POST_REMOTE**, respectively.

4.4.2 Enabling and Post Functions

The enabling and post functions are grouped into two classes: local and remote. The local functions are compiled into the player task itself, and are available to all applications. These functions are executed and immediately completed upon the firing of a transition.

The remote functions are independent of the controller, and require that some other task is available to perform the action. When a post function needs to be executed, the player sends a message to the appropriate servicer indicating that the function has been called. The player then continues with its normal execution sequence, leaving that particular transition in a firing state. Once the action is completed, the servicer sends a message to the player indicating that the transition can finish firing. This corresponds to the three transition firing phases discussed in Section 2.2.

4.4.3 Display File

The display configuration file provides the relationship between nodes in the display to nodes in the players. This allows the user to control the display content, either to reduce complexity or stress certain behaviors. Each place in the display Petri net is directly mapped to a place in a player Petri net. Transitions in the display are either mapped to a single transition or a pair of transitions. When two transitions are specified in the mapping, the firing of the first transition in the player starts the firing of the transition in the display. The transition's symbol is highlighted in the display while the transition is firing. When the second transition of the pair completes its firing, the transition in the display completes as well. This allows a subnet of the actual player Petri net to be represented as a single transition in the display. The configuration data for a display file starts with the line:

```
DISPLAY   task_name   primary_player
```

This identifies the file as a display configuration file, and includes the symbolic task name for this display and its primary player. The primary player indicates which player Petri net, if any, should be the recipient of the user runtime commands entered in this display that affect a single player. The format of a place consists of:

```
PLACE   local_name   remote_name remote_owner
```

The local_name is the name of the place that is displayed. The remote_name is the name of the place in the player Petri net which we want to display. The remote_owner is the task name of the player owning the remote place. The format of a transition is:

```
TRANS local_name  [SINGLE remote_name remote_owner]
                  |[DOUBLE start_name start_owner end_name end_owner]
```

SINGLE is used when there is a direct mapping for the transition, while DOUBLE is used when start firing corresponds to one transition and stop firing corresponds to a different transition, as discussed above.

4.5 User Interface

The user interface is implemented in the display Petri nets, and allows the user to control not only display features but the actual execution of the underlying Petri net. This interaction is provided by a combination of mouse and keyboard actions.

Table 4.1: Display Commands

Cause	Effect
Arrow Keys	Shift the Petri net in the direction of the arrow.
<,>	Scale the size of the Petri net down or up.
T,t	Toggle the tags on the nodes on or off.
R,r	Redraw the Petri net.
0,1, ... ,8,9	Toggle the enabling of the different display layers on or off. (Every element appears in layer 0)

4.5.1 Display Control

The user can manipulate the display in a number of ways for more efficient viewing of the application. These features are accessed through the keyboard, and are described in Table 4.1.

4.5.2 Player Control

The user can directly affect the execution of the underlying Petri net through the display. These actions are accessed through both the keyboard and the mouse. Table 4.2 describes the available commands.

4.6 Summary

This controller offers a simple, user friendly method of constructing and executing a distributed application. The interface to the external functions provides a flexible mechanism that will satisfy the needs of future applications of this controller. The user interface is also extremely flexible, both in terms of the actual display content and the display options.

Table 4.2: Player Commands

Cause	Effect
S	Set the entire application into single-step mode. An enabled transition will only fire when the user clicks on it with the mouse pointer.
s	Set the primary player for this display (as indicated in the display configuration file) into single-step mode. Other players are unaffected.
C	Set the entire application into continuous mode, where enabled transitions will fire as fast as possible.
c	Set the primary player for this display (as indicated in the display configuration file) into continuous mode. Other players are unaffected.
Selecting a transition by mouse	Fire the transition if it is enabled and the player is in single-step mode.
Selecting a place by mouse	Left Button: Add a token to the place. Middle Button: Prompts user for new token quantity. Right Button: Subtract a token from the place.
Q,q	Quit the application. User is prompted to verify.

CHAPTER 5
IMPLEMENTATION VS. THEORY

5.1 Introduction

Now that the controller has been presented, it is important to verify that the implementation maintains the desirable properties of Petri nets. There are two issues to be faced: 1) can a distributed Petri net be correctly analyzed with the existing tools; and 2) does the execution algorithm affect the properties of the net.

5.2 Analysis of Distributed Petri Nets

Ideally we would like to be able to analyze each Petri net in isolation, and from that infer the properties of the distributed system as a whole. If we can verify that the coordination structure maintains these properties, we can then develop and analyze each net independently.

This approach of analyzing the individual components to determine the global properties can be viewed as a synthesis problem. Can we begin with a single Petri net, and synthesize other connected Petri nets in such a fashion as to maintain the desired properties? If so, then the individual Petri nets in this connected structure can be constructed utilizing these synthesis techniques, and it can be shown that the global properties are unaffected.

This problem can be solved with a top-down synthesis technique, where an aggregate model is repeatedly refined [21]. This technique consists of refining a single transition with a well-formed block, shown in Figure 5.1. This transition must not be 2-enabled (the number of tokens in its input places is only sufficient for a single firing), and it is replaced with a well-formed block with these three conditions:

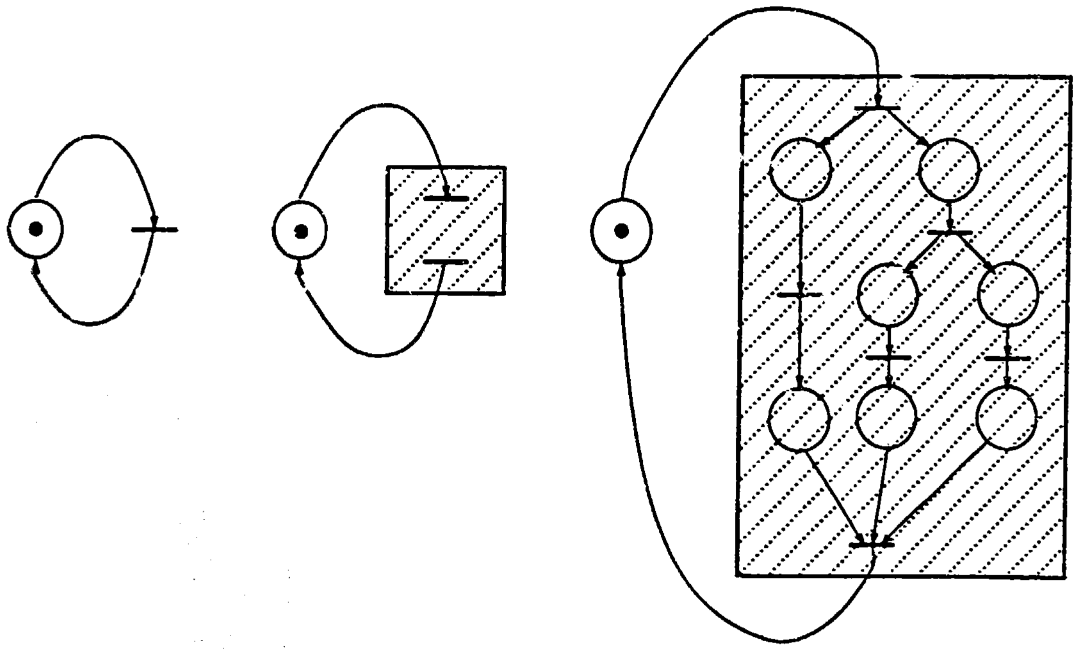


Figure 5.1: A Well-Formed Block

- The associated Petri net is live,
- The input transition to the block is only enabled when the associated net is idle,
- The input transition is the only transition enabled by the initial marking of the associated net.

Under these conditions, the liveness and boundedness of the overall net is preserved. The task here is to create a coordination structure for communication between two nets.

The coordination structure presented earlier does indeed meet the conditions for this synthesis technique.

- The coordination structure itself is live and bounded, and one of the previously discussed analysis tools can be used to verify whether the remainder of the

Petri net is live or not.

- The semaphore S ensures that the input transition **begin** is enabled only when the Petri net is idle.
- The use of places **ready** and S act to enable only **begin** under the initial marking. The initial marking of the remainder of the Petri net should not enable any other transitions.

Therefore, this coordination structure allows the isolated Petri nets to be verified as live and bounded independently of the other nets, and maintain those properties system wide.

There are a number of coordination structures that could be developed, but care must be taken to insure that they are valid. The coordination structure that was proposed by Wang [22] [25] in his development of the Petri Net Transducer contains a subtle error. This original coordination structure, along with the current version and a simplified implementation are shown in Figure 5.2. The original structure contained two semaphores, SI and SO. This mechanism was chosen to speed the execution of sequential requests. In theory, once an operation was completed, it could be communicating the results to the calling Petri net while another action was begun. However, this arrangement does not necessarily notify the correct transition of the completion of an operation, as the following scenario demonstrates.

Two separate **begin** - **end** pairs wish to request an action from a single Petri net. The input semaphore SI correctly allows only one of the **begin** transitions to fire. Once the first action is complete, transition **finish** will fire. This restores the input semaphore, and puts a token in place O. This is where the problem occurs. The second **begin** transition is now enabled and can fire. If it fires before the first **end** transition does, there will be a token in both **in_progress** places. This means that both **end** transitions are now enabled, and that either one could fire. If the first end fires, then there is no problem. However, it is possible for the second end to fire first, and this is a violation of the correct net structure, as the firing of the second end should correspond to the completion of the second operation, which is still in progress. This type of behavior is not only incorrect, but can be dangerous in a real system such as the CIRSSE testbed, where the controller is operating physically powerful and quick manipulators.

There are other possible valid coordination structures, as stated above. One such structure is shown at the bottom of Figure 5.2. This is a simplified (in terms of number of transitions and places) version of the current coordination structure. It removes a layer of transitions and places from the connection, but does not offer the intuitive structure of the current implementation. Future work includes the development of a coordination structure that allows a Petri net to perform parallel operations, while still maintaining the ability to analyze the net in isolation. A parallel implementation can greatly enhance the utilization of the underlying hardware, and reduce the time necessary to complete some global task. A good example occurs in the motion coordinator, where it would be desirable to allow both manipulators to move simultaneously, which is not possible with the current coordination structure.

5.3 Execution Algorithm

It has been shown that a distributed Petri net maintains the desirable properties of a single net. Therefore, we wish to examine the behavior of the actual execution algorithm in reference to the properties of Petri nets. It will be shown that the chosen algorithm does not fully uphold the property of fairness, as well as demonstrate that fairness is not strictly necessary for a Petri net controller.

The execution algorithm consists of randomly selecting a transition to examine, and firing it if enabled. Otherwise, progress linearly through the transition list,

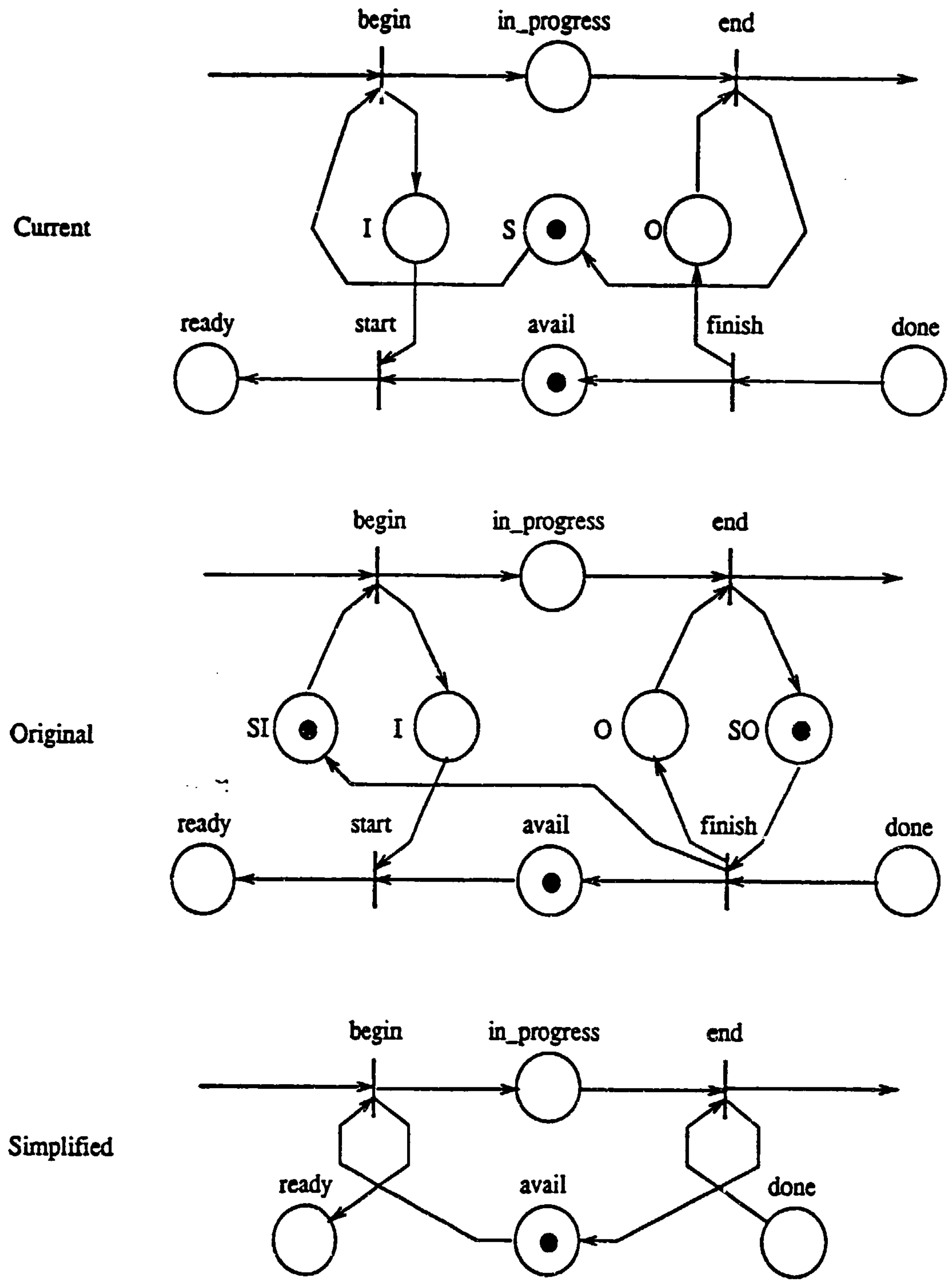


Figure 5.2: A Coordination Structure Comparison

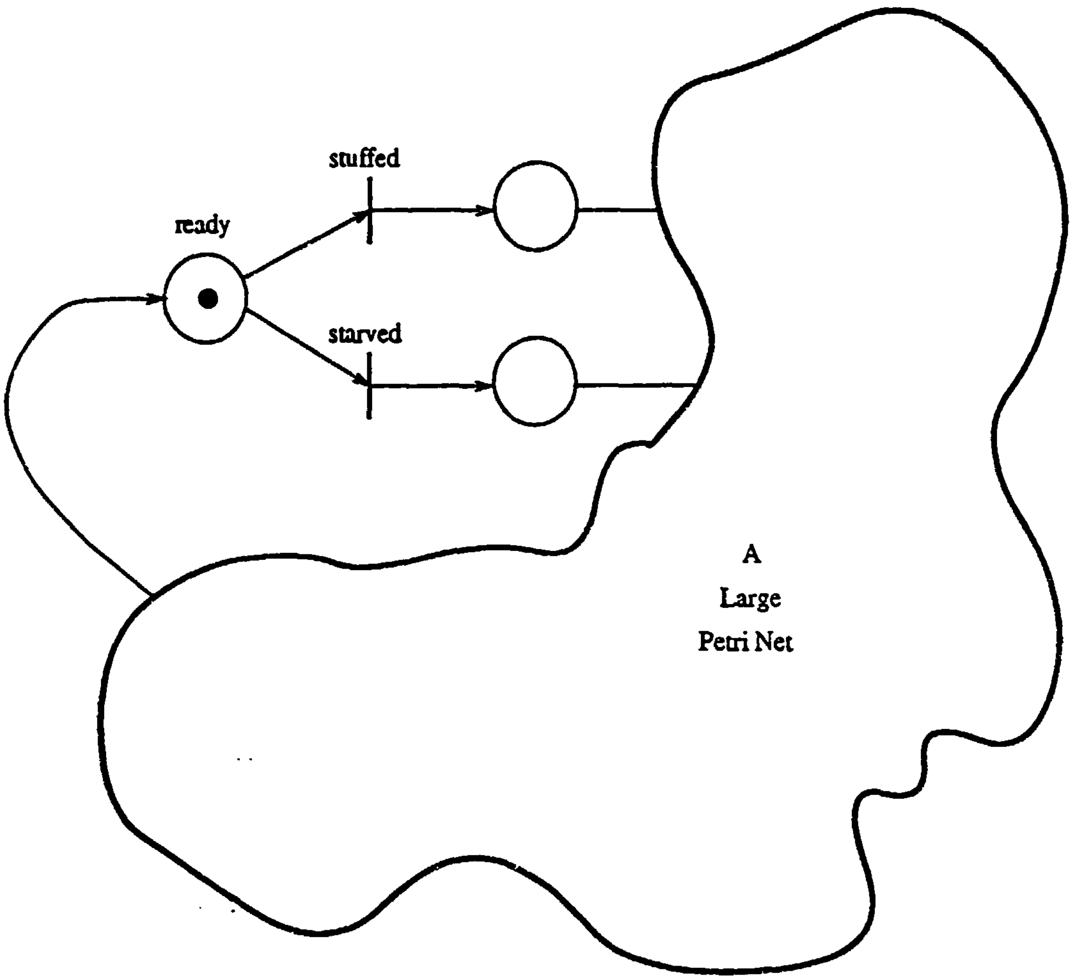


Figure 5.3: An Unfair Example

checking each transition until one can be fired. The case of concern is shown in Figure 5.3, where two transitions are enabled by the choice place ready. Suppose that the transition starved occurs immediately after transition stuffed in the internal data structure for the controller. Both of these transitions are enabled when there is a token in ready. Following the execution algorithm, a random transition is selected for examination. Since stuffed is always enabled when starved is, and it occurs immediately before starved in the data structure, it will be fired first unless **starved** is selected directly by the first random choice. This can be significant if the remainder of the net has a large number of transitions. Taken to the extreme, if only these two transitions are enabled in a large net with thousands of transitions, then **stuffed** will be fired thousands of times more often than **starved**. Fortunately, this behavior is very uncommon in controller applications. Due to the distributed nature of many systems, the individual Petri nets tend to be small and manageable. Secondly, very few control algorithms have a true choice place. The common transitions will typically have an enabling function or tape requirement to control the final selection of which transition to fire.

5.4 Summary

Through the use of a consistent interconnection scheme, distributed Petri nets can be analyzed in isolation, and provide information about the system as a whole. Additionally, the implementation presented here maintains all of the desirable properties of Petri nets that are relevant to controller applications.

CHAPTER 6
CASE STUDY: THE CIRSSE TESTBED

6.1 Introduction

This chapter describes the architecture of the CIRSSE dual arm robotic testbed, and presents two case studies demonstrating the integration of multiple subsystems through the use of this controller.

6.2 The Dual Arm Robotic Testbed

The Theory of Intelligent Machines [18] [20] presents a hierarchical structure for the organization of an autonomous robotic system. This hierarchy is based on the principle of Increasing Precision with Decreasing Intelligence. There are three distinct levels to this hierarchy: the organization level which provides for global task planning; the coordination level which serves to integrate the physical capabilities of the system and decompose global plans into a sequence of lower-level operations; and the execution level which consists of the actual hardware and low level functional capabilities.

This structure has found an implementation in the CIRSSE dual arm robotic testbed [9] [26]. The execution level is segmented into two subsystems: the Motion Control System (MCS) and the Vision Services System (VSS). MCS is responsible for interfacing with the low level capabilities of the manipulators. The hardware consists of two 6 degree-of-freedom PUMA robotic arms each mounted on a 3 degree-of-freedom cart which moves along a common linear track. This provides for a total of 18 degrees-of-freedom. The MCS software is executed on a dedicated VME cage, running the VxWorks operating system. VSS is responsible for interfacing to the cameras and image processing hardware. Five cameras may be used for stereo or mono vision purposes; two are mounted to the ceiling, two are mounted on the end-effector of one robot, and one which is not yet dedicated. A laser scanner is available for pointing or pattern projection. These devices are connected to a Datacube image processing system residing in a VME cage.

The coordination level is of primary interest to this thesis, as the Petri net controller resides here. The coordination level receives commands from the organization level, and decomposes them into portions manageable at the execution level. The coordination level contains two interior levels, consisting of a dispatcher and a number of coordinators.

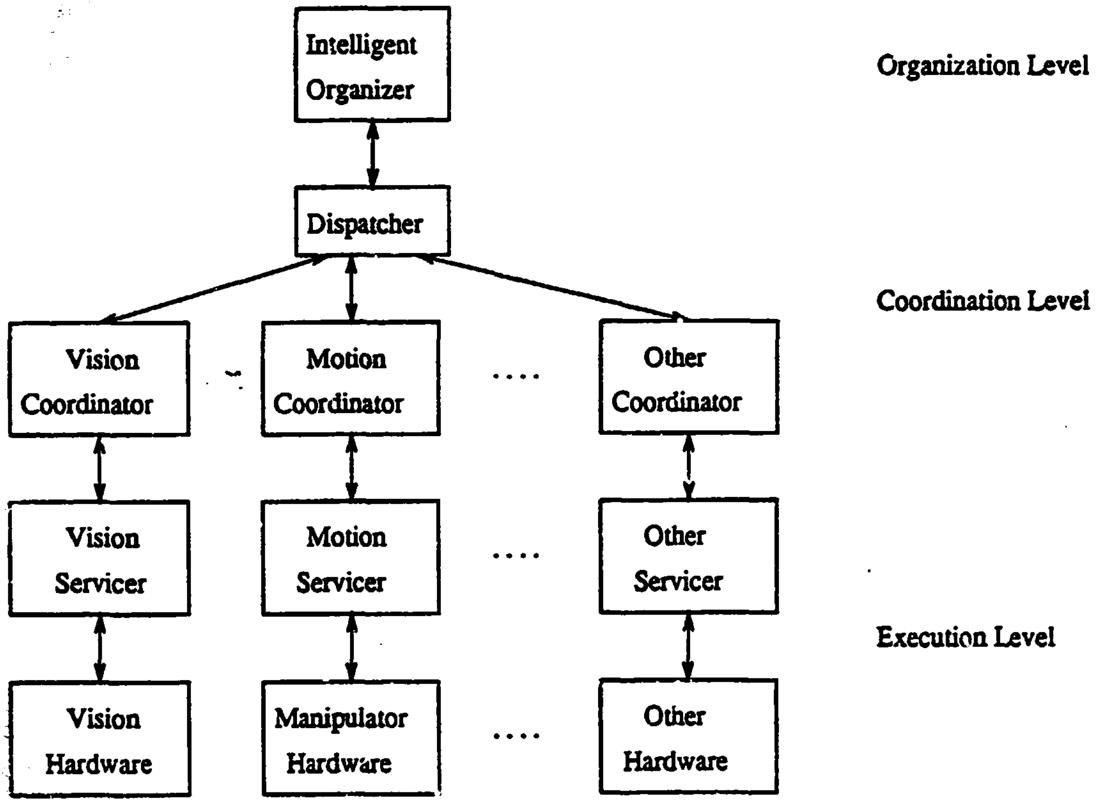


Figure 6.1: The Hierarchy of an Intelligent Machine

The hierarchical configuration of an intelligent machine is shown in Figure 6.1, where it is apparent that the coordination level is also arranged hierarchically. The dispatcher and coordinators are all player tasks in the controller. The coordinators represent the capabilities of their particular subsystem, while the dispatcher serves to decompose high-level commands into lower-level commands suitable for the individual coordinators. Another feature to note is that the coordinators are designed to process only one action at a time from the dispatcher. This does not preclude a particular coordinator from performing concurrent operations in order to process a particular command.

6.3 Case 0: Camera Calibration

Before the testbed can be fully utilized, the ceiling cameras must be calibrated. The algorithm for this process is simple, yet it makes full use of the key concepts developed for the controller. A light source is placed in the open gripper of one of the manipulators. The manipulator grasps the light, and moves it to a fixed number of positions in the viewing region of the ceiling cameras. At each position the vision system determines the position of the light in terms of camera coordinates, while the motion system determines the position of the light in terms of world coordinates. These values are stored in a file, and can be accessed as needed by the various components. Once all of the data points have been obtained, the camera calibration parameters are calculated, thereby completing the camera calibration process.

This distributed system has been analyzed using the Petri net tool GreatSPN and the method presented in Section 5.2, and was found to be live and bounded. Each Petri net was analyzed in isolation, and their individual properties of liveness and boundedness can be applied to the system as a whole due to the use of coordination structures for inter-net communication.

6.3.1 Dispatcher

The dispatcher serves to parse high-level commands into lower-level commands

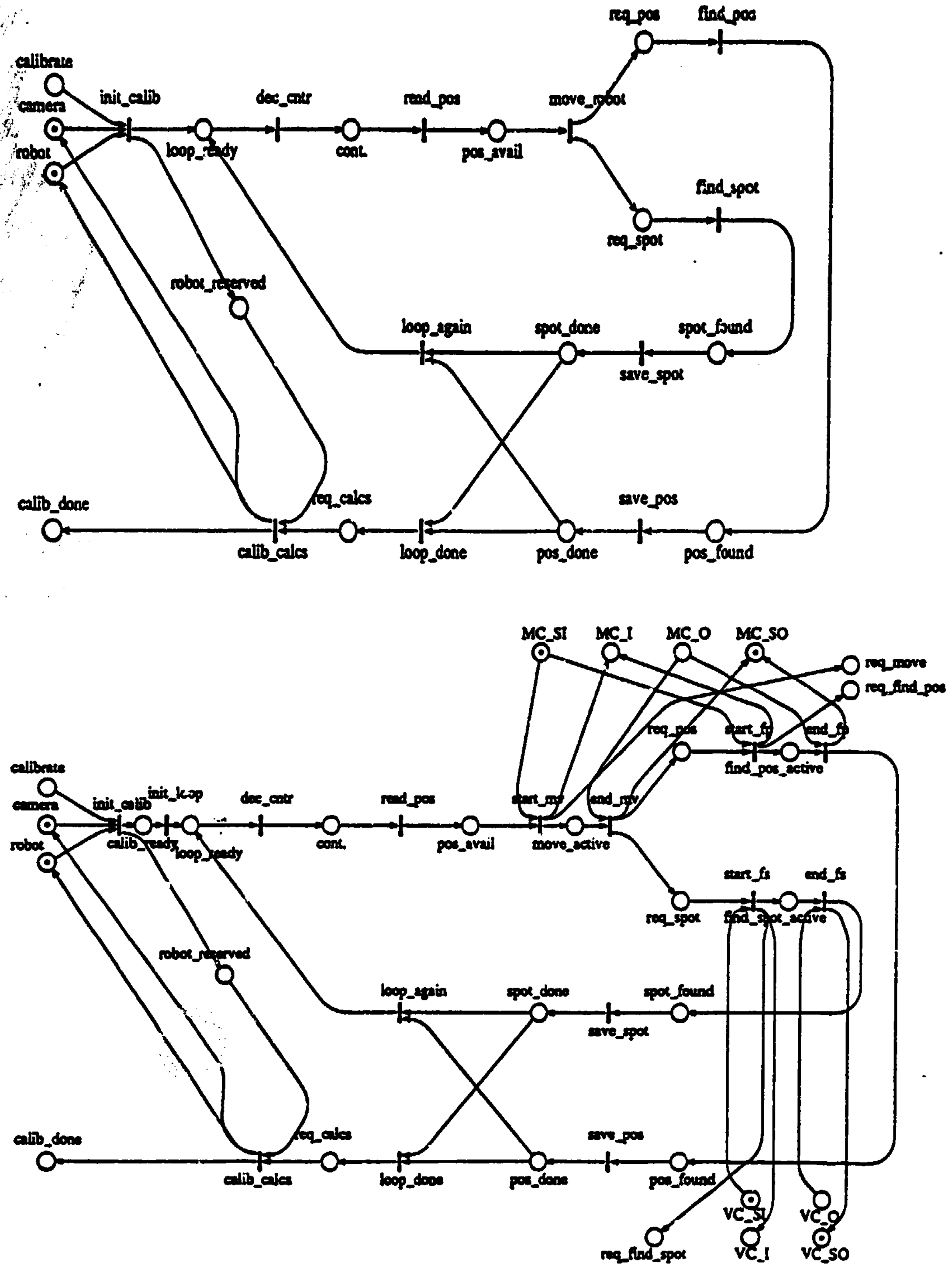


Figure 6.2: Case 0: Dispatcher Display and Player

understandable by the individual coordinators. In this case, the dispatcher performs the overall algorithm, while the coordinators control their respective systems. The display and player Petri nets for the dispatcher are shown in Figure 6.2, which are the upper and lower Petri nets, respectively. Here you can see that the Petri net shown on the display is a slightly simplified version of the underlying net, omitting the communications details necessary for interfacing with the coordinators.

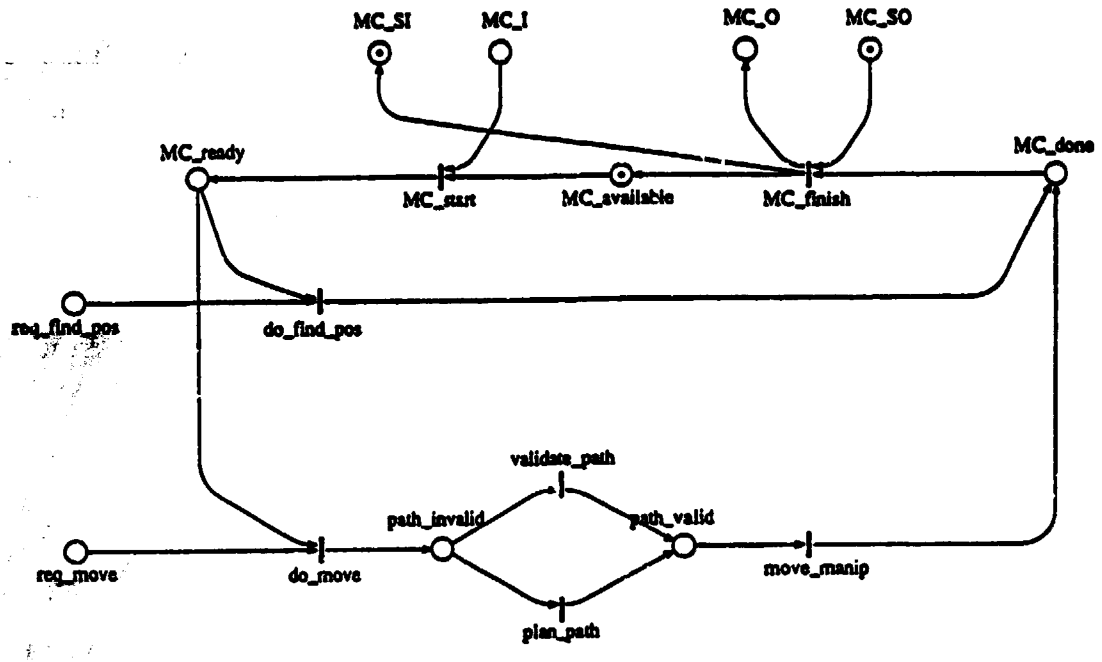
The command **calibrate_camera** is given to the dispatcher by placing a token in the place **calibrate**. This enables the transition **init_calib** if the robot and camera are available. The transition **init_calib** in the display represents the two initialization transitions in the player, namely **init_calib** and **init_loop**. **Init_calib** in the player executes the remote post function which opens the file containing the calibration points. **Init_loop** executes a local post function which initializes a loop variable to the number of calibration points. The transitions **dec_cntr** and **read_pos** decrement the loop variable and load the next calibration point from the file, respectively. The transition **move_robot** corresponds to a coordination structure which requests the motion coordinator to move to the destination position. This transition does not complete its firing until the motion coordinator replies that it is done. The net now splits into two parallel paths, demonstrating the ability to perform concurrent operations. The transition **find_pos** is another coordination structure which requests the motion coordinator to find the exact location of the robot, while **save_pos** stores this value in a file. The transition **find_spot** is a coordination structure which requests the vision coordinator to locate the bright spot in the cameras field of view, and **save_spot** stores this value in a file. This process of moving and storing location information is repeated a number of times corresponding to the loop variable. The transition pair **loop_again** and **loop_done** perform the actual looping action. An enabling function is attached to each transition, which checks the status of the looping variable. If the loop variable is now zero, then loop_done is enabled, otherwise loop_again is enabled. After all of the calibration data has been obtained, the loop is exited and calib_calcs fires. A post function performs the calculation of the camera calibration parameters, and the resources are returned.

6.3.2 Motion Coordinator

The coordinators reflect the capabilities of their particular subsystem, hiding many of the details from the dispatcher. The motion coordinator is very simple for this case, possessing only two actions callable by the dispatcher. The display and player Petri nets are shown in Figure 6.3, and you can see that both Petri nets are identical. The transition do_find_pos queries the motion servicer for the position of the manipulator used for the calibration. The other transition do_move activates the portion of the net that is responsible for moving the robot. Two paths exist within this action, depending upon whether a path plan already exists or not. If the path already exists (and for this case it does, as the dispatcher is simply executing moves to preplanned robot joint positions) we can proceed to moving the manipulator via validate_path and move_manip. Otherwise, the motion coordinator would fire plan_path to generate a path, and then move_manip to move the manipulator.

6.3.3 Vision Coordinator

The vision coordinator for this case represents the minimum Petri net structure for a coordinator. The display and player Petri nets are shown in Figure 6.4, and like the motion coordinator, they are identical. There is only one action callable by the dispatcher, and is activated by transition do_find_spot which locates the light spot in the camera's field of view and stores its position in the DOM.



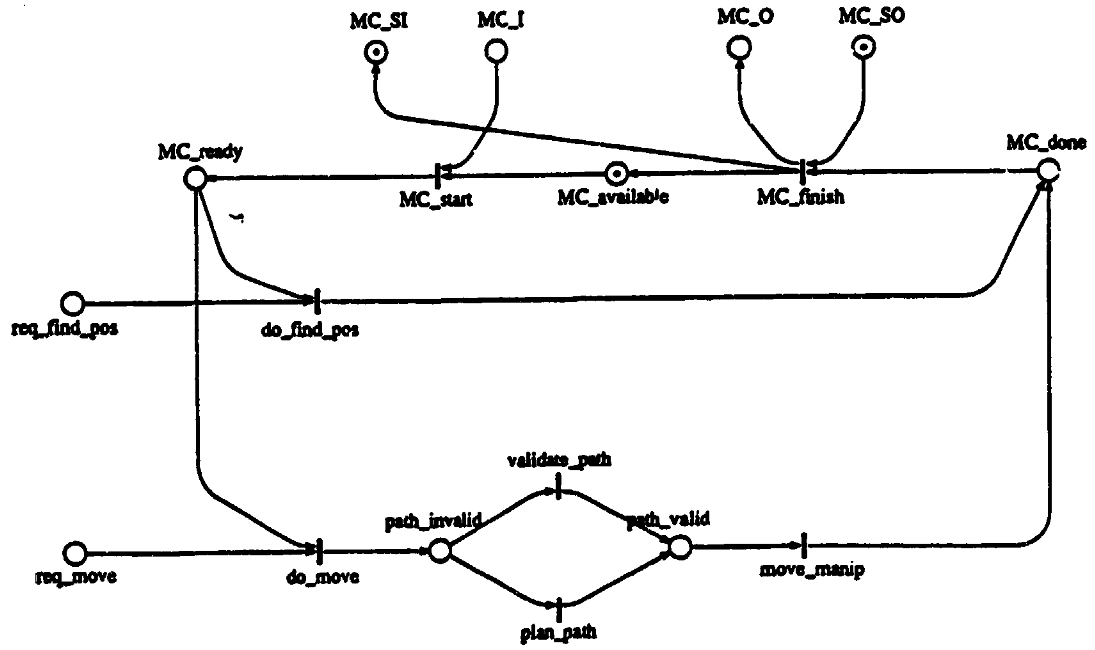
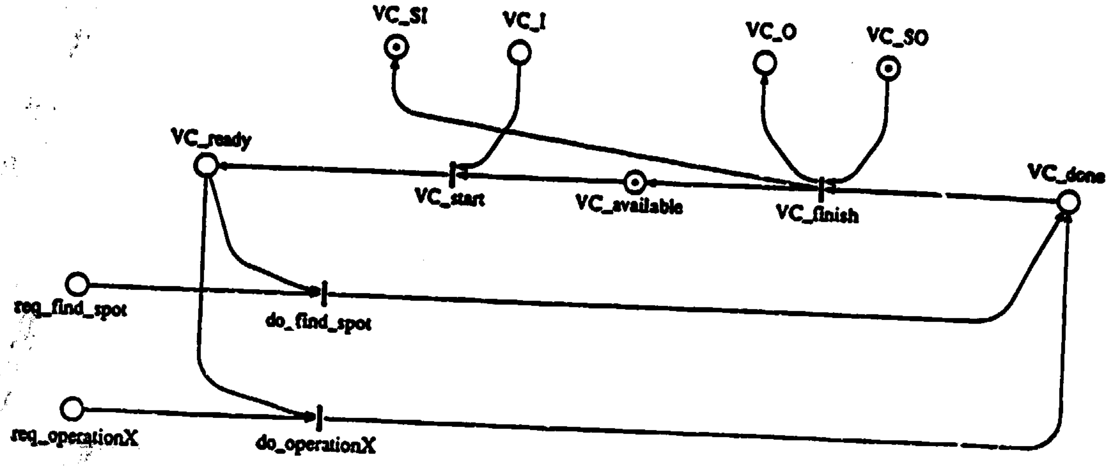


Figure 6.3: Case 0: Motion Coordinator Display and Player



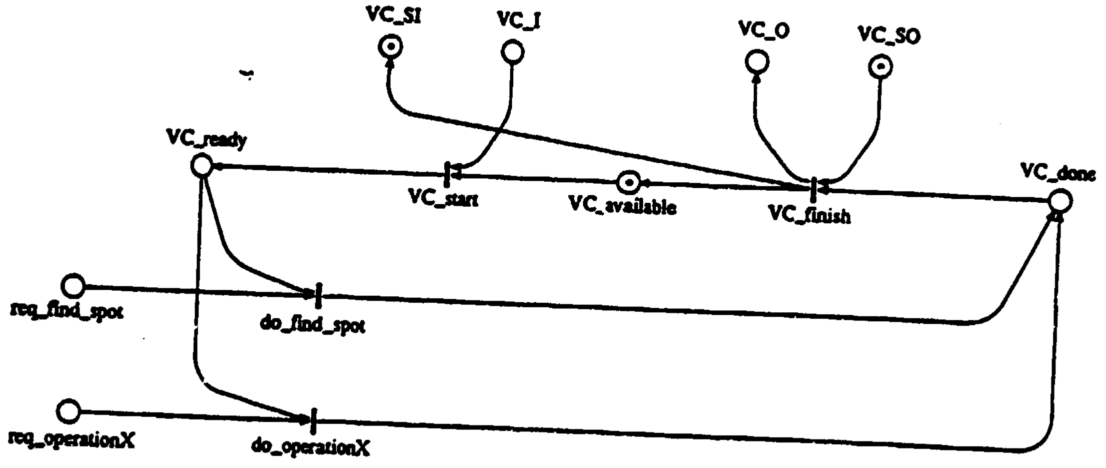


Figure 6.4: Case 0: Vision Coordinator Display and Player

6.4 Case 1: Strut Insertion

This case considers a more difficult task which is more representative of a typical use of the robotic platform. A partially completed strut and node triangle is placed on a table, and the task at hand is the insertion of the third strut to complete the triangle. This task is significantly more complicated than the camera calibration, with about an order of magnitude more transitions and places. This problem is shown in Figure 6.5.

This case study has not been fully executed yet, as the functional code for the path planner is currently under development. However, the controller is being used for the testing and debugging of those modules still under construction.

6.4.1 Dispatcher

The dispatcher for this application makes good use of the controllers ability to perform parallel operations, as well as to coordinate the sequencing of asynchronous events. Due to the size and complexity of this net a transition by transition explanation will not be given. Rather, a verbal overview of the dispatcher's actions will be presented. The Petri nets for the display and player tasks are shown in Figures 6.6 and 6.7, respectively.

The triangle completion begins by initializing the vision system in parallel with the manipulator and path planning systems. Note that the path planner cannot begin initialization until the robot is initialized, since it requires knowledge of the position of the robot and must move it to the home position if it is not already there.

The vision system must now locate the partially completed triangle, and calculate the approximate position of the first uncompleted node. The path planner plans a path to this node, and the motion system moves the manipulator to it. While the arm is in motion, the path planner is already calculating the path to the

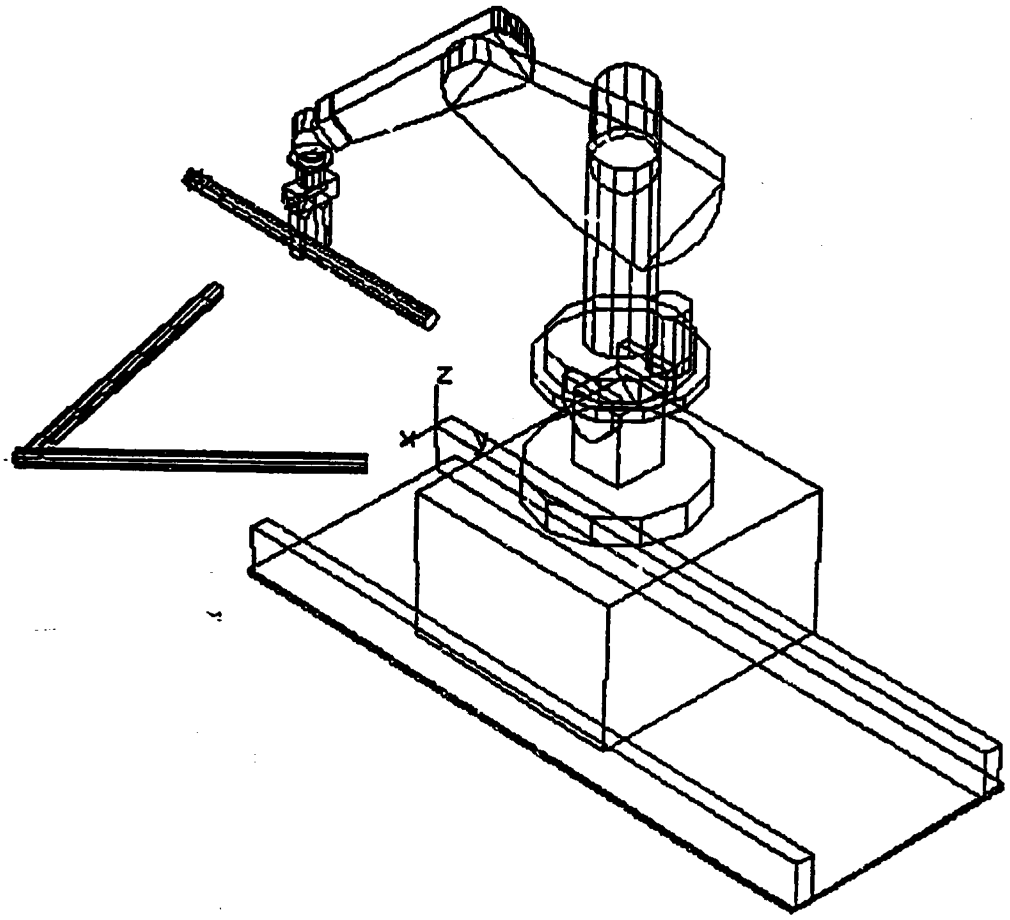


Figure 6.5: Case Study 1: Strut Insertion

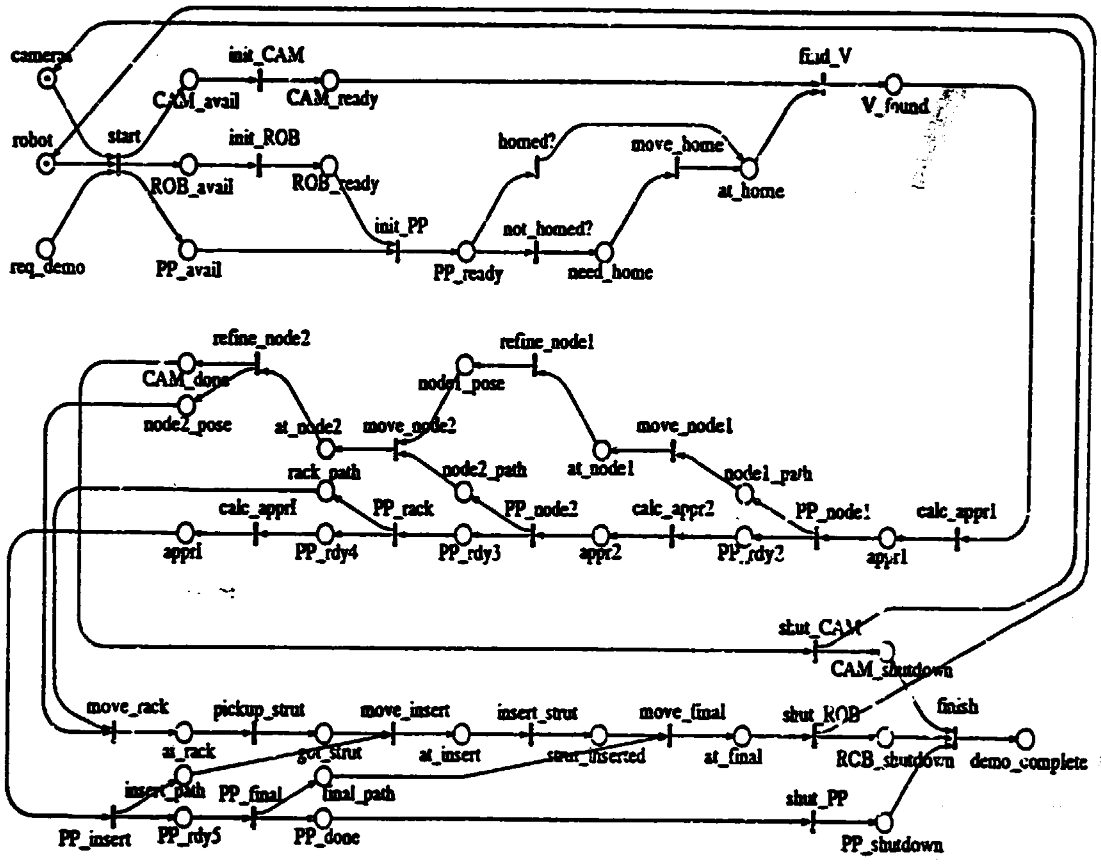


Figure 6.6: Case 1: Dispatcher Display

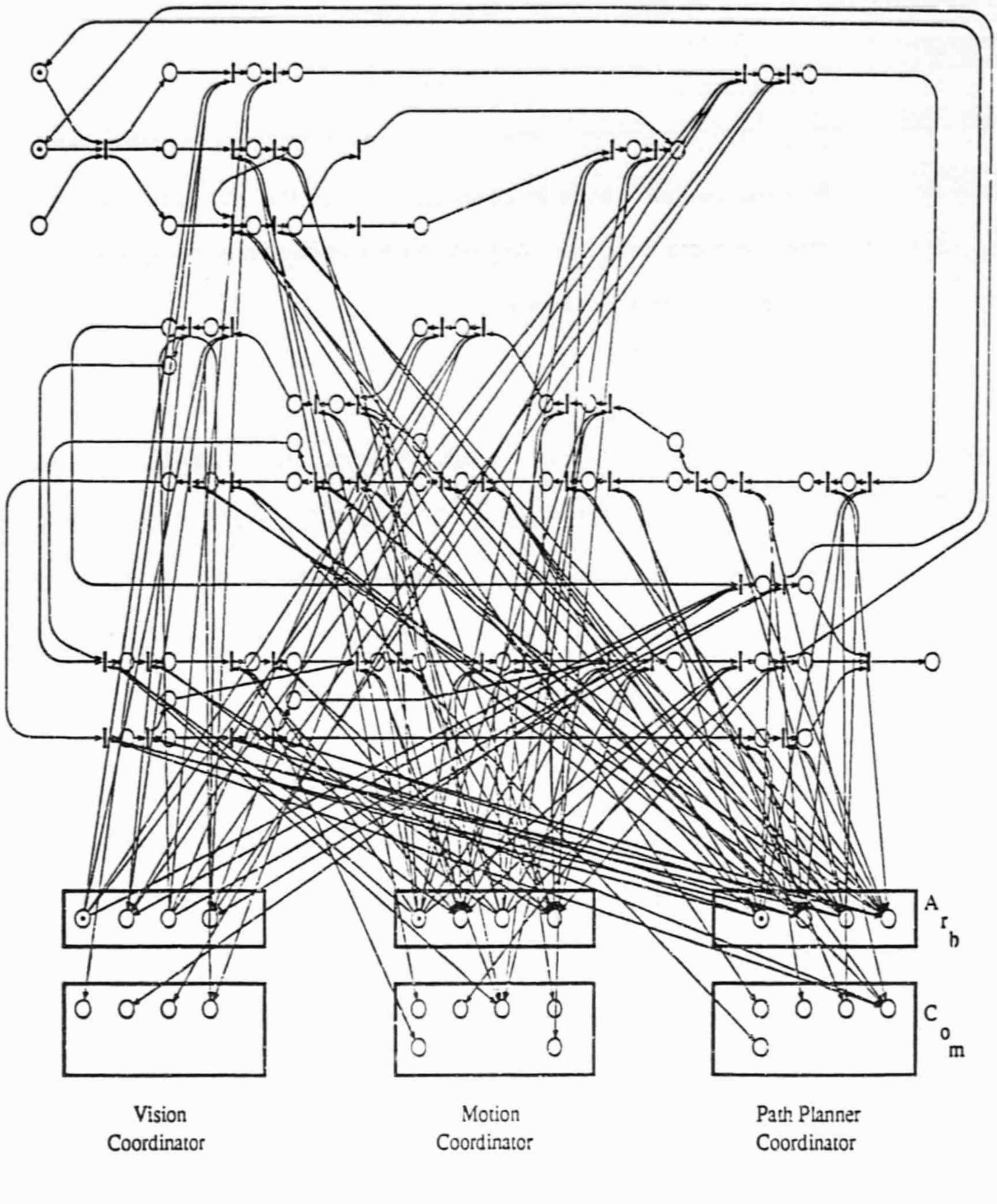


Figure 6.7: Case 1: Dispatcher Player; Note the Complexity versus the Dispatcher Display Petri Net

second node. The camera's on the arm now locate the node more accurately, and the arm moves to the second node where its location is refined as well.

While the location of the second node is being refined, the path planner is calculating a path to the rack containing extra struts. Once the arm is available, it moves to the rack and picks up a strut. During this time, the path planner is first calculating a path to insert the strut in the uncompleted triangle, and then to move to some final safe position. The arm moves to an approach position over the triangle, and inserts the strut.

The triangle is now complete, so the robot moves to the safe position, and the three subsystems are shut down.

The dispatcher for this case provides a good example of why such a flexible display system was developed. The algorithm given above is clear in the display Petri net, where each action to be performed by the coordinators is represented by a single transition. The coordination structures in the player Petri net make it very difficult to understand, with a large number of arcs crossing each other as well as the individual places and transitions. The coordination structures for the three coordinators are indicated in the boxed regions at the bottom of the figure. The upper boxes labeled *Arb* correspond to the Arbitration portion of the structures, where access to the particular coordinator is controlled. The lower boxes labeled *Com* correspond to the different commands offered by each coordinator. Note that the tags for the places and transitions are not shown in the player Petri net, since they were illegible.

6.4.2 Motion Coordinator

The player Petri net for the motion coordinator is shown in Figure 6.8, and it contains seven operations. The transitions **do_init_robot** and **do_shutdown_robot** are used to initialize and shutdown the robot. The transition **do_find_pos** determines the exact position of the robot, as known by sensors on the manipulator itself. The last four transitions perform different types of move commands, ranging from a general move to a specific sequence such as inserting a strut into a node pair.

6.4.3 Vision Coordinator

The player Petri net for the vision coordinator is shown in Figure 6.9, and it performs five operations. Similarly to the motion coordinator, two of these transitions, **do_init_camera** and **do_shutdown_camera**, are used to initialize and shutdown the camera system. The transitions **do_find_spot** and **do_find_object** locate the bright spots and a particular object, respectively. The last transition, **do_refine_pose**, more accurately determines the location of a node through the use of the arm cameras. This requires that the node must be located relative to the arm cameras, and that the position of the cameras be known. The location of the node in world coordinates is determined after these two parallel operations complete.

6.4.4 Path Planner Coordinator

The path planner coordinator is slightly different from the other coordinators in that it has no direct execution level counterpart. The path planner is strictly a software module, and is not responsible for interfacing to any hardware.

The player Petri net for the path planner is shown in Figure 6.10, and it consists of five operations. Two transitions, **do_init_PP** and **do_shutdown_PP**, are responsible for the initialization and shutdown of the path planner. The remaining transitions calculate different kinds of paths, ranging from a simple start-to-end point path, to specific paths such as a current position to approach position for insertion path.

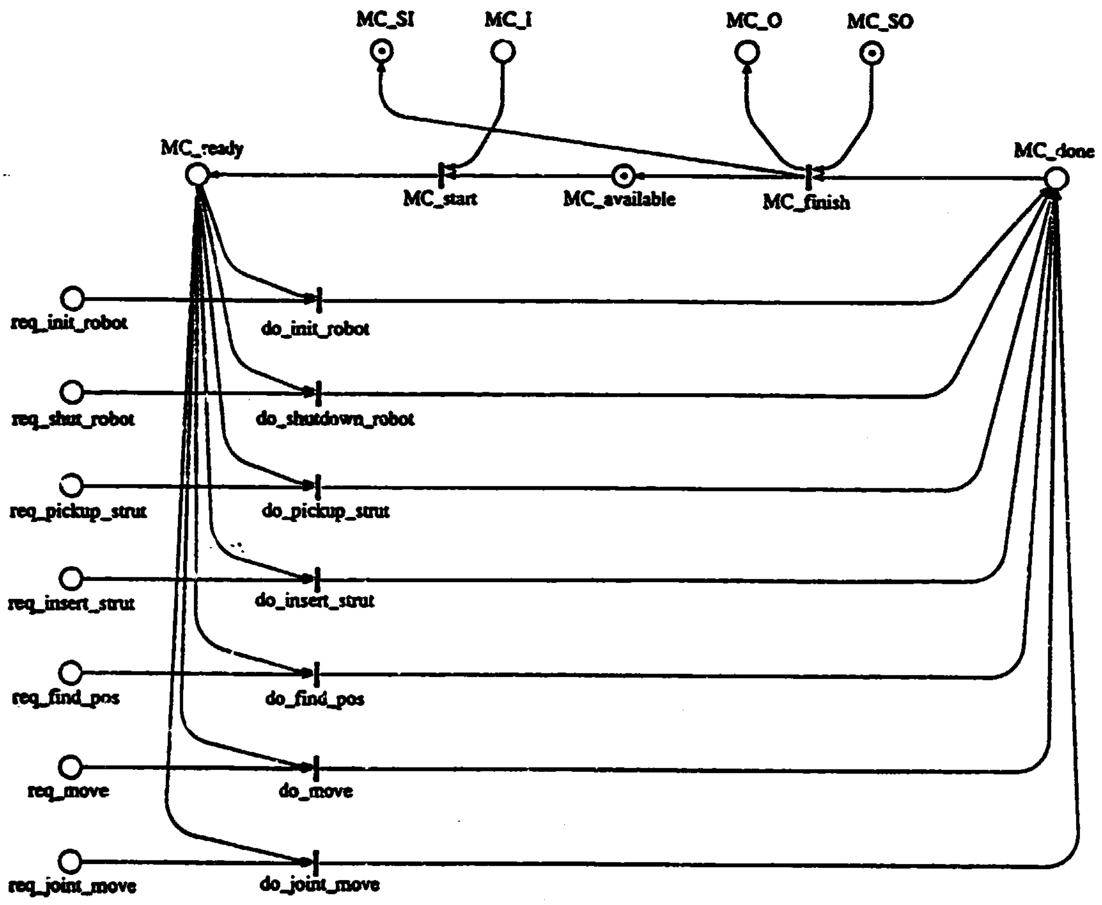


Figure 6.8: Case 1: Motion Coordinator Player

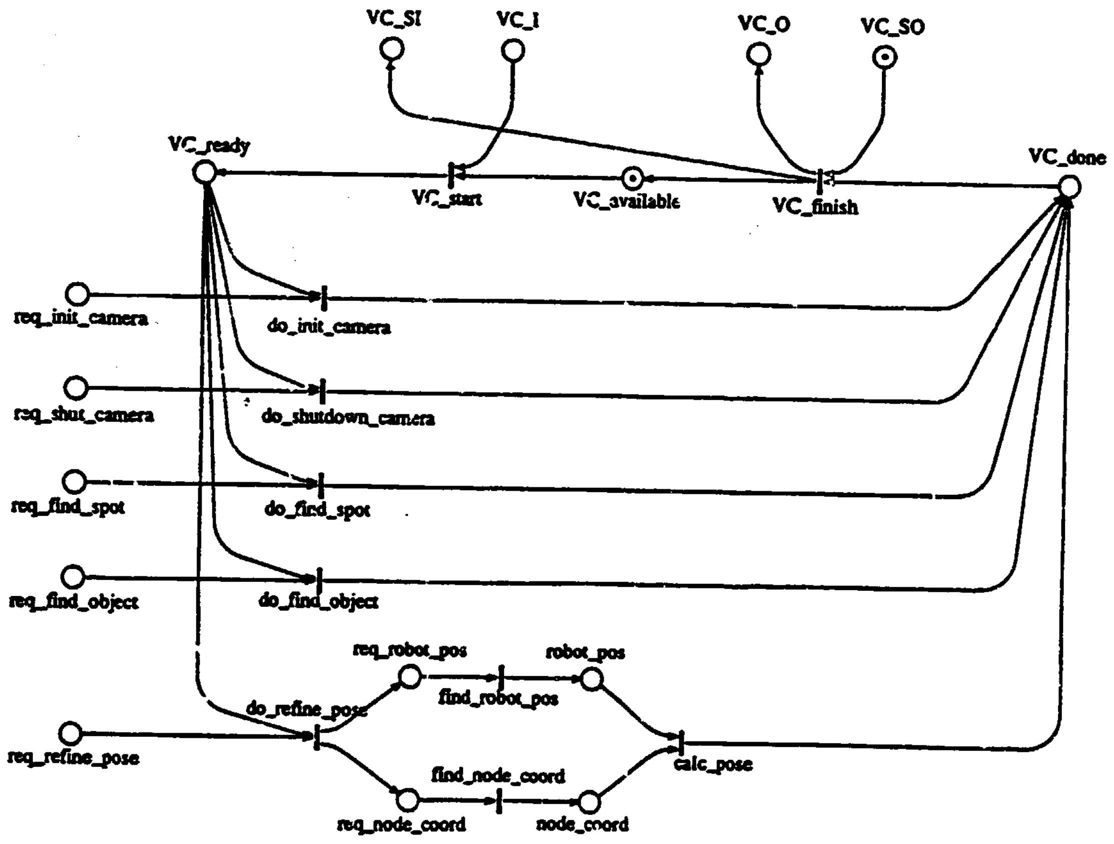


Figure 6.9: Case 1: Vision Coordinator Player

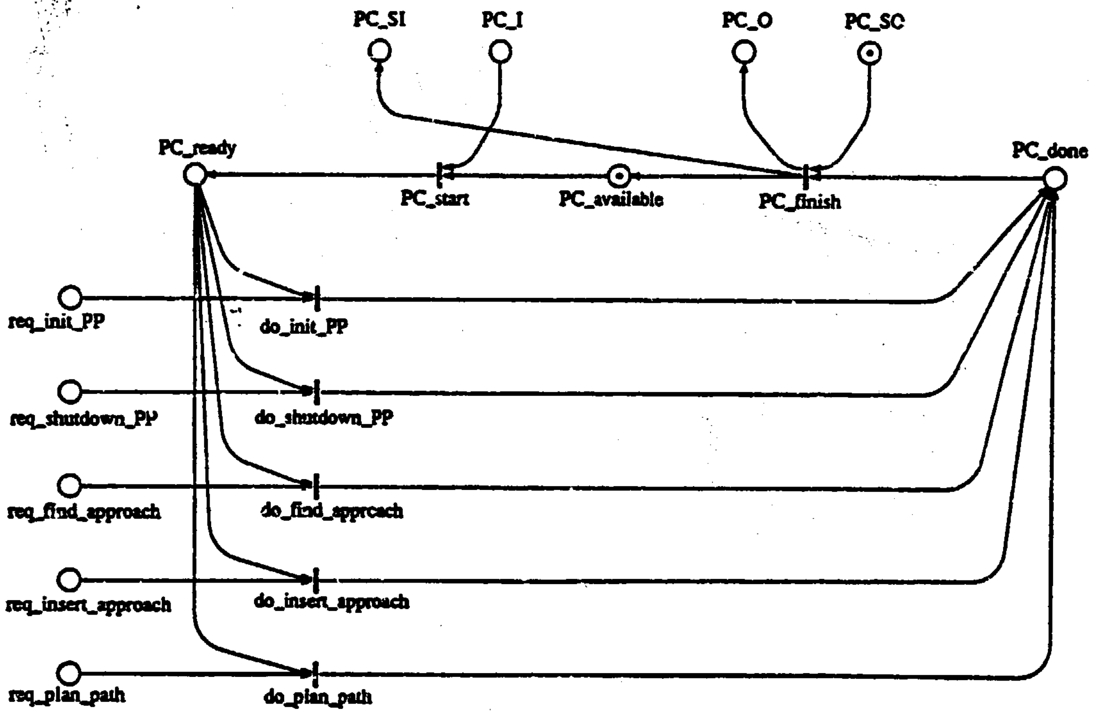


Figure 6.10: Case 1: Path Planner Coordinator Player

6.5 Summary

The capabilities of this controller have been demonstrated through two case studies on the CIRSSE dual arm robotic testbed. Case study 0 has been implemented and executed, successfully providing a method for the simple control over calibrating the stereo ceiling cameras. The functional code for case study 1 is still under development, and the controller is providing a user friendly method of testing the individual components as they are constructed.

CHAPTER 7
CONCLUSIONS AND FUTURE RESEARCH

7.1 Conclusions

This thesis described the need for a tool which could provide some means of modeling, integrating and controlling an application. A Petri net based controller was presented as a solution, and two case studies demonstrated that it could satisfy these criteria for a distributed heterogeneous hierarchical system. Its main strengths are:

- A controller algorithm based on Petri nets which allows the system to be mathematically analyzed for performance and desirable properties such as liveness, boundedness, etc.
- Straightforward design and interconnectivity of the individual Petri nets.
- Compatible with CTOS
  - Fully integrated with the Execution-level services provided by MCS and VSS.
  - Easily configurable for distributed and heterogeneous applications.
- Demonstrated parallelism of the CIRSSE testbed
  - Modeled and controlled a hierarchical configuration.
  - Provided for the operation of a multiprocessor system.
- A user friendly graphical interface for the monitoring and control of each Petri net.

- Highly configurable displays for the more efficient representation of the underlying Petri net control structure.

7.2 Future Research and Modifications

- Expand the firing of transitions to record performance data. This information can be used to enhance the analysis of the net with one of the previously mentioned tools. Remember, those modeling tools require that the user enter his best guess for timing parameters. Initial estimates can be made and then enhanced as the system is implemented for more refined modeling.
- Supplement keyboard commands with equivalent on-screen X-window gadgets. Many operations, such as scrolling the display, would be more efficient with a mouse and gadget interface.
- Construct an event handler task for editing and sending Petri Net Transducer tapes to the player tasks.
- Provide for outlining or otherwise marking sections of the Petri net in the displays. This would allow for visible segmentation of associated groups within the Petri net.
- Develop or enhance existing Petri net design tools such as Xpn or GreatSPN to fully support the creation and modification of the player and display configuration files.

LITERATURE CITED

[1] Al-Jaar, R.Y., Desrochers, A.A., "Petri Nets in Automation and Manufacturing", *Advances in Automation and Robotics*, (G.N. Saridis, ed.), vol. 2, pp. 153-225, JAI Press, Greenwich, CT, 1990.

[2] Bjanes, A., "A Distributed Petri Net Controller for a Dual Arm Testbed", *Master's Thesis*, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, May 1991.

[3] Chiola, G., "GreatSPN 1.5 Software Architecture", University of Torino, July 1990.

[4] Ciardo, G., "Manual for the SPNP Package Version 3.0", Duke University, June 1990.

[5] Crockett, D., "Manufacturing Workstation Control Using Petri Nets", *Master's Thesis*, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, August 1986.

[6] Fieldhouse, K., Holt, K., Lefebvre, D., Murphy, S., Swift, D., Watson, J., "Lecture Materials for the CTOS/MCS Introductory Course", *CIRSSE Report 97*, Rensselaer Polytechnic Institute, July 1991.

[7] Freedman, P., "Time, Petri Nets, and Robotics", *IEEE Transactions on Robotics and Automation*, vol. 7, no. 4, pp. 417-435, Aug 1991.

[8] Kim, J., "Xpn: Petri Net GUI", Rensselaer Polytechnic Institute, Oct. 1991.

[9] Lefebvre, D.R., Saridis, G.N., "A Computer Architecture for Intelligent Machines", *CIRSSE Report 108*, Rensselaer Polytechnic Institute, 1991.

[10] Marsan, A., "Stochastic Petri Nets: An Elementary Introduction", *Advances in Petri Nets 1989* (G. Rozenberg, ed.), pp. 1-29, Springer-Verlag, 1989.

[11] Mittmann, M., "TokenPasser: A Petri Net Specification Tool", *Master's Thesis*, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, May 1991.

[12] Murata, T., "Petri Nets: Properties, Analysis and Application", *Proceedings of the IEEE*, vol. 77, no.4, pp. 541-580, April 1989.

[13] Peck, J., "GreatSPN2SPNP User's Manual", *CIRSSE Unix Man Pages*, Rensselaer Polytechnic Institute, August 1990.

[14] Peterson, J.L., "Petri Net Theory and the Modeling of Systems", Prentice-Hall International, Englewood Cliffs, NJ, 1981.

[15] Robinson, J., "Performance Analysis of a Robotic Testbed Control Architecture", *Master's Thesis*, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, December 1989.

[16] Rudolph, D., "Petri Net-Based Control of a Flexible Manufacturing System", *Master's Thesis*, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, May 1989.

[17] Saridis, G.N., Stephanou, H.E., "A Hierarchical Approach to the Control of A Prosthetic Arm", *IEEE Trans. Sys., Man and Cyber.*, vol. 7, pp. 407-420, 1977.

[18] Saridis, G.N., "Architectures for Intelligent Machines", *CIRSSE Report 96*, Rensselaer Polytechnic Institute, Aug 1991.

[19] Saridis, G.N., "Intelligent Machines: Distributed vs. Hierarchical Intelligence", *Proc. IFAC/IMAC Int. Symp. on Distrib. Intelligence Systems* (Varna, Bulgaria), pp. 34-39, 1988.

[20] Saridis, G.N., "Toward the Realization of Intelligent Controls", *Proceedings of the IEEE*, vol. 67, no. 8, 1979.

[21] Valette, R., "Analysis of Petri Nets by Stepwise Refinement", *Journal of Computer Systems Science*, vol. 18, pp 35-46, 1979.

[22] Wang, F., "A Coordination Theory for Intelligent Machines" *Doctoral Thesis*, Electrical, Computer, and Systems Engineering Department, Rensselaer Polytechnic Institute, May 1990.

[23] Wang, F., Saridis, G.N., "A Formal Model for Coordination of Intelligent Machines using Petri Nets", *Proc 3rd IEEE Int. Intell. Control Symp.* (Arlington, VA), 1988.

[24] Wang, F., Kyriakopoulos, K.J., Tsolkas, A., Saridis, G.N., "A Petri-Net Coordination Model of Intelligent Mobile Robots", *CIRSSE Report 50*, Rensselaer Polytechnic Institute, Jan. 1990.

[25] Wang, F., Saridis, G.N., "Structural Formulation of Plan Generation for Intelligent Machines", *CIRSSE Report 40*, Rensselaer Polytechnic Institute, Sept. 1991.

[26] Watson III, J.F., Lefebvre, D.R., Desrochers, A.A., Murphy, S.H., Fieldhouse, K.R., "Testbed for Cooperative Robotic Manipulators", in *Intelligent Robotic Systems For Space Exploration* , (A.A. Desrochers, ed.), pp. 1-38, Kluwer Academic Press, Norwell, MA, 1992.

APPENDIX A
Local Enabling and Post Functions

The local enabling and post functions were initially implemented to provide a means of performing deterministic looping behavior. They have been expanded to include any functions that are relatively small, generic to many applications, and linked to the player code.

A.1 *Enabling Functions*

```
done  action  data_name
        Use:  To check the state of a local variable.
        Returns: TRUE if the contents of data_name = 0,
                 FALSE otherwise.
                 This behavior holds when action = 1,
                 if action = -1 the logic is reversed.


DOM_done  action  data_name
        Use:  To check the state of a variable in  the DOM.
        Returns: TRUE if the contents of data_name = 0,
                 FALSE otherwise.
                 This behavior holds when action = 1,
                 if action = -1 the logic is reversed.


DOM_bool  action  data_name
        Use:  To check the state of a variable in  the DOM.
        Returns: FALSE if the contents of data_name = 0,
                 TRUE otherwise.
```

This behavior holds when action = 1,

if action = -1 the logic is reversed.

A.2 *Post Functions*

initialize action data_name

Use: To initialize a local variable to a fixed value.

Operation: Sets the contents of data_name to action.

live_init action data_name

Use: To initialize a local variable to a runtime value.

Operation: Prompts the user to enter the desired value,
and sets the variable data_name.

decrement action data_name

Use: To decrement a local variable by a fixed value.

Operation: Subtracts action from the contents of data_name.

increment action data_name

Use: To increment a local variable by a fixed value.

Operation: Adds action to the contents of data_name.

renameObjs action data_name

Use: To rename a variable in the DOM.

Operation: Here data_name is the comma delimited source
and destination variable names.

dupObjs action data_name

Use: To duplicate a variable in the DOM.

Operation: Here data_name is the comma delimited source and destination variable names.

alive action data_name

Use: To print a flag when a transition fires.

Operation: Prints the string "alive: " followed by the integer value of action.

APPENDIX B
Header Files

B.1 *Global_var.h* — The Global Variables

```
/* CTOS variables */

   TID_TYPE globTid;

/* X-display variables */

   Display *display;
   Window  *window;
   GC      gc;
   XFontStruct *font[4], *act_font;
   int     screen_num;
   char    display_name[100];

/* Size variables */

   int     diameter, radius, radius2;
   float   scale, offset_x, offset_y;
   unsigned int line_width;

/* user options */

   int mult, tags_on, rates_on, visible_layer = 0xffff;
   int mode = SINGLE_STEP;
   TID_TYPE playTid;

/* net objects */

   struct Place *place;
   struct Trans *trans;
   struct MPar *mpar;
   struct RPar *rpar;

   int num_mp, num_pl, num_rp, num_tr, num_gr, num_cs;

   char  *net_name ;

/* tape variables */

struct tape_node *head, *tail;

struct tape_entry tape_table[] =
   {{1,"MOVE"},
    {2,"GRASP"},
    {3,"LOOK"},
    {4,"STOP"},
    {5,"CAL_VIS"}
   };
```

B.2 *Global_var.p* — The Global Variable Prototypes

```
/* CTOS variables */

   extern TID_TYPE globTid;

/* X-display variables */

#include <X11/Xlib.h>

   extern Display *display;
   extern Window  *window;
   extern GC       gc;
   extern XFontStruct *font[4], *act_font;
   extern int      screen_num;
   extern char     display_name[100];

/* Size variables */

   extern int      diameter, radius, radius2;
   extern float    scale, offset_x, offset_y;
   extern unsigned int line_width;

/* user options */

   extern int mult, tags_on, rates_on, visible_layer;
   extern int mode;
   extern TID_TYPE playTid;

/* net objects */

   extern struct Place *place;
   extern struct Trans *trans;
   extern struct MPar *mpar;
   extern struct RPar *rpar;

   extern int num_mp, num_pl, num_rp, num_tr, num_gr, num_cs;

   extern char  *net_name ;


/* tape variables */

   extern struct tape_node *head, *tail;

   extern struct tape_entry tape_table[];
```

B.3 *Global_def.h* — The Global Definitions

```
/*
  FILE:  Global_def.h

*/

/* general defines */

#define MAXLINE 255

/* modes of operation */
```

```
#define CONTINUOUS  1
#define SINGLE_STEP 2

/* transition ID's */

#define EXP  0
#define IMM  1
#define DET  127
#define DEFAULT 200

/* arc ID's */

#define INPARC 0
#define OUTARC 1
#define INHARC 2

/* type of config file */

#define  CF_PNET  (1)                    /* config file for subnet player  */
#define  CF_DISP  (2)                    /* config file for subnet display */

/* type of net node */

#define  PLACE_NODE  (0)
#define  TRANS_NODE  (1)

/* graphics defines */

#define  PIX_PER_IN 60
#define  SCALE 1
#define  DIAMETER 23
#define  IN_TO_PIX(a) (int)((a)*PIX_PER_IN)
#define  X_IN_TO_PIX(a) (int)(IN_TO_PIX(a)*scale+IN_TO_PIX(offset_x))
#define  Y_IN_TO_PIX(a) (int)(IN_TO_PIX(a)*scale+IN_TO_PIX(offset_y))
#define  ANG (M_PI/12)
#define  max(a,b) (((a)>(b))? a:b)
#define  ERASE 0
#define  DRAW 1
#define  INVERT 2
#define  S 0.4
#define  M 0.5
#define  L 0.7

/* net elements */

#include <ctos.h>

typedef  unsigned int     UINT ;
typedef  unsigned short   USHORT ;

#define  T_START  0x01
#define  T_END    0x02
#define  T_BOTH   0x03

typedef struct _updlst
    {
    TID_TYPE         tid ;
    USHORT           num ;
    USHORT           which;    /* start or end firing of trans */
    struct _updlst   *next ;
    }
```

```
UPDATES ;

struct Arrowhead
   {int    x0,y0,x1,y1,x2,y2;
   };

struct Arc_point
   {int    x,y;
    float  x_in, y_in;
    struct Arc_point *next;
   };

struct Arc
   {int    x_start, y_start;
    float  x_start_in, y_start_in;
    struct Arc_point *arc_points;
    int    x_end, y_end;
    float  x_end_in, y_end_in;
    int    type;
    int    mult;
    int    place;
    int    layer;
    struct Arrowhead *arrow;
    struct Arc *next;
   };

struct Place
   {char       *name;
    UINT        id ;
    int         orig_token;
    int         curr_token;
    int         x_loc, y_loc;
    float       x_loc_in, y_loc_in;
    int         tag_x, tag_y;
    float       tag_x_in, tag_y_in;
    int         layer;
    int         locked;
    int         tid;
    TID_TYPE    owner ;
    char       *act_name;
    UPDATES    *update ;
   };

struct Trans
   {char       *name;
    char       *act_name;
    char       *act_end_name;
    UINT        id ;
    UINT        end_id ;
    float       fire_rate;
    int         kind;
    int         orient;
    int         x_loc, y_loc;
    float       x_loc_in, y_loc_in;
    int         tag_x, tag_y;
    float       tag_x_in, tag_y_in;
    int         rate_x, rate_y;
    float       rate_x_in, rate_y_in;
    struct Arc *inp_arc;
    struct Arc *out_arc;
    struct Arc *inh_arc;
    int         layer;
    int         firing;
    int         tape_req;
    int         enable_type;
    int        (*enable_func)();
    int         enable_action;
    int         enable_var;
    char       *enable_data;
    int         post_type;
    void       (*post_func)();
    TID_TYPE    post_tid;
    int         post_action;
    int         post_var;
    char       *post_data;
    TID_TYPE    owner ;
    TID_TYPE    end_owner ;
    UPDATES    *update ;
   };

struct MPar
   {char   *name;
    int    num_token;
    int    tag_x, tag_y;
    float  tag_x_in, tag_y_in;
    int    layer;
   };

struct RPar
   {char   *name;
    float  fire_rate;
    int    tag_x, tag_y;
    float  tag_x_in, tag_y_in;
    int    layer;
   };


/*  struct for registering P & T  */
typedef struct
    {
    char  *sym_name [64] ;
    char  *act_name [64] ;
    int    type ;
    int    id ;
    USHORT which;
    }
REG_REMOTE ;


/* Tape stuff */

#define MAX_TAPE 5

struct tape_node
   {int id;
    struct tape_node *next;
   };

struct tape_entry
   {int id;
    char *ident_str;
   };
```

```
/* enable and post function stuff */

#define F_NONE   0
#define F_LOCAL  1
#define F_REMOTE 2

#define REMOTE_POST_FUNC "send_post_message"

struct func_entry
   {int (*func_ptr)();
    char *ident_str;
   };

#define  OBJ_NAME_LENGTH  256

typedef struct
    {
    int     indx ;
    int     action ;
    char    data_obj [OBJ_NAME_LENGTH] ;
    }
POST_DATA;


/* variable stuff */

#define MAX_VAR 200

struct var_entry
   {int   value;
    char *ident_str;
   };

/*  macro definitions  */

#define  MAKE_ID(tid,type,indx)  (  (((tid )<<16) & 0xffff0000)    \
  | (((type)<<15) & 0x00008000)    \
  | ( (indx)      & 0x00007fff))

#define  TID_FROM_ID(id)         ( ((id) & 0xffff0000) >> 16 )
#define  TYPE_FROM_ID(id)        ( ((id) & 0x00008000) >> 15 )
#define  INDEX_FROM_ID(id)       (  (id) & 0x00007fff        )

#define  ENCODE_BUTTON(b,id)     ( (((b)<<16) & 0xffff0000) \
                                  |( (id)     & 0x0000ffff))
#define  GET_BUTTON(id)          ( ((id) & 0xffff0000) >> 16)
#define  GET_INDEX(id)           (  (id) & 0x0000ffff       )

#define  LEFT    0x00000001
#define  RIGHT   0x00000002
#define  UP      0x00000004
#define  DOWN    0x00000008

#define  HORI_SHIFT(id)          ( (((id) & LEFT) * -1)  \
                                  + (((id) & RIGHT) >> 1) )
#define  VERT_SHIFT(id)          ( ((((id) & UP) >> 2) * -1)  \
                                  + (((id) & DOWN) >> 3) )

#define  SCALE_FACTOR   0.1
#define  DEC            -1
```

```
#define   INC            1
```

B.4 *msg.h* — The Controller CTOS Messages

```
/* Message commands */
#define MSG_COORD              11000

#define MSG_DRAW_WHOLE_NET    MSG_COORD+1
#define MSG_SET_TOK           MSG_COORD+2
#define MSG_START_TRANS       MSG_COORD+3
#define MSG_END_TRANS         MSG_COORD+4
#define MSG_APP_TAPE          MSG_COORD+5
#define MSG_INS_TAPE          MSG_COORD+6
#define MSG_CLR_TAPE          MSG_COORD+7
#define MSG_ABS_TAPE          MSG_COORD+8
#define MSG_ADD_TOKEN         MSG_COORD+9
#define MSG_SUB_TOKEN         MSG_COORD+10
#define MSG_CHK_FIRE_TRANS    MSG_COORD+11
#define MSG_TRANS_FIRED       MSG_COORD+12
#define MSG_TRANS_DONE        MSG_COORD+13

/* Mouse Actions */
#define MSG_SELECT_TRANS      MSG_COORD+20
#define MSG_SELECT_PLACE      MSG_COORD+21

/* Keyboard Actions */
#define MSG_CHANGE_SCALE      MSG_COORD+30
#define MSG_SHIFT_DISPLAY     MSG_COORD+31
#define MSG_TOGGLE_TAGS       MSG_COORD+32
#define MSG_SET_MODE          MSG_COORD+33
#define MSG_VISIBLE           MSG_COORD+34

#define MSG_SIT_ON_HANDS      MSG_COORD+40

#define MSG_YOU_ARE           MSG_COORD+51
#define MSG_INIT_DISPLAY      MSG_COORD+52
#define MSG_INIT_PLAYER       MSG_COORD+53
#define MSG_REGISTER_REMOTE   MSG_COORD+54
#define MSG_CONFIRM_REMOTE    MSG_COORD+55
#define MSG_REGISTER_UPDATE   MSG_COORD+56
```

APPENDIX C
Player Software

C.1 *makefile* — The Player Makefile

```
# Makefile for test of UNIX version of CTOS

INC1 = /home/page/MCS/src/lib/btsLib
INC2 = /home/page/MCS/src/lib/msgLib
INC3 = /home/page/MCS/src/lib/ctosShell
INC4 = /usr2/testbed/CIRSSE/installed/UNIX/h
INC5 = /home/page/MCS/src/lib/recLib
INC6 = /home/lefebvre/unix/ctos/domLib
INC7 = /home/lefebvre/DEMOS/demo1

CFLAGS = -O -Wreturn-type -Wunused -Wswitch -Wcomment -Wshadow
    -Wpointer-arith -DMAKE_SUN4 -I$(INC1) -I$(INC2) -I$(INC3)
    -I$(INC4) -I$(INC5) -I$(INC6) -I$(INC7) -DOS_UNIX

BINDIR = /home/page/MCS/bin/sun4/
BTSLIB = $(BINDIR)btsErr.o $(BINDIR)btsSem.o $(BINDIR)btsGlobals.o
         $(BINDIR)msgLib.o $(BINDIR)msgBCMem.o
RECDIR = /home/page/MCS/bin/sun4/

# List of targets...
# ----------------

all : play_eh


# E.H. Tasks...
# -----------

play_eh:  play_eh.o Xplay.o load_net.o tape.o msgs.o post_func.o \
 enable_func.o var_man.o
gcc $(CFLAGS) -o play_eh  play_eh.o  Xplay.o  tape.o  msgs.o \
post_func.o  enable_func.o load_net.o var_man.o $(RECDIR)recLib.o \
$(BTSLIB) $(BINDIR)ctosShell.o \
-L/home/lefebvre/DEMOS/demo1 -L/home/lefebvre/unix/ctos/domLib \
-lcoord -ldom -lm


play_eh.o: play_eh.c Global_var.h Global_def.h
gcc -c $(CFLAGS) play_eh.c -o play_eh.o

Xplay.o: Xplay.c Global_def.h
gcc -c $(CFLAGS) Xplay.c -o Xplay.o

load_net.o: load_net.c Global_def.h
gcc -c $(CFLAGS) load_net.c -o load_net.o

tape.o: tape.c Global_def.h
gcc -c $(CFLAGS) tape.c -o tape.o

post_func.o: post_func.c Global_def.h
gcc -c $(CFLAGS) post_func.c -o post_func.o
```

```
enable_func.o: enable_func.c Global_def.h
gcc -c $(CFLAGS) enable_func.c -o enable_func.o

msgs.o: msgs.c Global_def.h
gcc -c $(CFLAGS) msgs.c -o msgs.o

var_man.o: var_man.c Global_def.h
gcc -c $(CFLAGS) var_man.c -o var_man.o
```

C.2 *play_eh.c* — The Player Event Handler

```
/*********************************************************************************
    File:     play_eh.c

    Purpose:  "Petri Net Player" event handler task executes subnet

    Author:   Joe Peck
              revised:  Don Lefebvre

    Revised:  19 Aug'91
*********************************************************************************/
/*------------------------------------ include files  ------------------*/
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

#include <stdio.h>

#include <ctos.h>
#include <btsLib.h>

#include "recLib.h"
#include "msg.h"
#include "Global_def.h"
#include "Global_var.h"
#include "Xplay.p"

/*------------------------------------ definitions  -------------------*/

/*------------------------------------ global variables  ---------------*/
int     go = FALSE;
char    root_name[40];
char    *myName ;
char    config_file [MAXLINE] ;

/*------------------------------------ function prototypes  ------------*/
STATUS  load_net        (char *net_config_file, int type) ;
STATUS  register_remote () ;
STATUS  confirm_remote  (TID_TYPE req, REG_REMOTE *ps) ;
STATUS  save_remote     (REG_REMOTE *ps) ;
STATUS  save_update     (MSG_TYPE *m) ;
STATUS  initial_marking () ;
void    printPTinfo     () ;
int     cntArcs         (struct Arc *p) ;

int     find_place      (char *name) ;
int     find_trans      (char *name) ;

MyTask(play_eh)
```

```
/*****************************************************************************
  play_eh  - subnet player for Petri Net
------------------------------------------------------------------------------
  SYMBOLIC NAME:  (used to configure net)

  RETURNS:        0 indicating processing complete

  PARAMETERS:     TID_TYPE  myTid - task id of event handler
                  MSG_TYPE *msg   - pointer to received message
*****************************************************************************/
int  play_eh (TID_TYPE myTid, MSG_TYPE *msg)
    {
    int  t_num ;


    switch (msg->command)
        {
        case MSG_TASK_ARGS:
             strcpy(root_name, (char *)msg->data);
             break;

case MSG_PINIT:
/*           setRunTime (120) ;  */

     /*  read display config file  */
             globTid = myTid;
     myName = msgMyTaskName (myTid) ;
/*             sprintf (config_file, "%s%s", dir_name, myName) ; */
             if (load_net (root_name, CF_PNET) == ERROR)
        {
 recInfo (myTid, "Unable to load config file '%s.pnet'",
  root_name);
        msgApplicationExit (myTid) ;
 break ;
        }
             /* recInfo (myTid, "PLAY: Loaded net '%s.pnet'\n", root_name); */

             break ;


case MSG_AINIT:
     register_remote () ;
             break ;


        case MSG_AEXEC:
     initial_marking () ;
             go = TRUE;
     /* printPTinfo () ;  */
             break ;


        case MSG_REGISTER_REMOTE:
     confirm_remote (msg->source,
     (REG_REMOTE *) msg->data) ;
             break;


        case MSG_CONFIRM_REMOTE:
                save_remote ((REG_REMOTE *) msg->data) :
                        break;


                   case MSG_REGISTER_UPDATE:
                save_update (msg) ;
                        break;


                   case MSG_APP_TAPE:
                        recInfo(myTid,"PLAY: Received APP_TAPE\n");
                        append_tape((int)msg->data);
                        break;

                   case MSG_INS_TAPE:
                        recInfo(myTid,"PLAY: Received INS_TAPE\n");
                        insert_tape((int)msg->data);
                        break;

                   case MSG_CLR_TAPE:
                        recInfo(myTid,"PLAY: Received CLR_TAPE\n");
                        clear_tape();
                        break;

                   case MSG_ABS_TAPE:
                        recInfo(myTid,"PLAY: Received ABS_TAPE\n");
                        absorb_tape();
                        break;


                   case MSG_TRANS_DONE:
                        /* recInfo (myTid, "%s: Received TRANS_DONE\n", myName) ; */
                        fire_end ((int)msg->data) ;
                        break ;


                   case MSG_SET_TOK:
           /*             recInfo (myTid, "%s: Received SET_TOK\n", myName) ; */
                        set_token((int)msg->data);
                        break ;

                   case MSG_ADD_TOKEN:
           /*             recInfo (myTid, "%s: Received ADD_TOKEN\n", myName) ; */
                        add_token ((int)msg->data) ;
                        break ;

                   case MSG_SUB_TOKEN:
           /*             recInfo (myTid, "%s: Received SUB_TOKEN\n", myName) ; */
                        sub_token ((int)msg->data) ;
                        break ;

                   case MSG_CHK_FIRE_TRANS:
           /*             recInfo (myTid, "%s: Received CHK_FIRE_TRANS\n", myName) ; */
                        if (go == TRUE)
                           {if (trans_tok_enabled((int)msg->data))
                               {if (trans_pre_enabled((int)msg->data))
                                   {/* recInfo(myTid,"%s: Firing transition %d\n",myName,
                                    (int)msg->data); */
                                    fire_start((int)msg->data);
                            if (done_firing((int)msg->data))
                                       fire_end((int)msg->data);
                        };
                    };
                };
                break ;

        case MSG_SET_MODE:
                mode = (int)msg->data;
                break;

        case MSG_ATERM:
                go = FALSE;
                break;
}


    /*  execute subnet if no msgs are waiting  */
    if ((go == TRUE)&&(mode == CONTINUOUS))
        {
while (msgQueueCount(myTid) == 0)
    {
    t_num = get_ready_trans();
    if (t_num == -1)
        {
usleep (500) ;
        continue ;
        }

    /* recInfo(myTid, "\n%s: ***  Firing Trans '%s' [%i]  ***\n",
    myName, trans[t_num].name, t_num) ; */
    fire_start(t_num);

    if (done_firing(t_num))
        fire_end(t_num);
    }
        }

    /*  execute subnet if no msgs are waiting  */
    if ((go == TRUE)&&(mode == SINGLE_STEP))
        {
        /* recInfo(myTid,"%s checking hidden transitions\n",myName); */
 while (msgQueueCount(myTid) == 0)
    {
    t_num = get_ready_hidden_trans();
    if (t_num == -1)
        {
usleep (500) ;
        continue ;
        }

/*            recInfo(globTid,"Firing hidden trans %s\n",
                                 trans[t_num].name); */
            fire_start(t_num);
            if (done_firing(t_num))
         fire_end(t_num);
    }
        }

    /*  exit event handler task via default processing  */
    return (msgDefaultProc (myTid, msg)) ;
    }
```

```
/*******************************************************************************
  get_updates  - returns update list as char string of tid's
*******************************************************************************/
char  *get_updates (UPDATES *ps)
    {
    static char  buf [80] ;
    char        *p ;

    *buf = '\0' ;
    p = buf ;
    while (ps)
        {
sprintf (p, "  %4x:%i", ps->tid, ps->num) ;
p = buf + strlen(buf) ;
ps = ps->next ;
        }

    return (buf) ;
    }


/*******************************************************************************
  printPTinfo  - print place & transition information
*******************************************************************************/
void  printPTinfo ()
    {
    int  i ;

    if (!strncasecmp (myName, "PNET_A", 6))
        sleep (1) ;
    else if (!strncasecmp (myName, "PNET_B", 6))
        sleep (2) ;

    recInfo (globTid, "\nNet Name: %s\n", myName) ;

    recInfo (globTid, "PLACES [%i]\n", num_pl) ;
    for (i=0; i<num_pl; i++)
        {
/* recInfo (globTid, "\t%6s  %4.1f  %4.1f  %2i\n", place[i].name,
 place[i].x_loc_in, place[i].y_loc_in, place[i].orig_token) ;*/
recInfo (globTid, "\t%12s  %8x  %4x  %s\n", place[i].name,
 place[i].id, place[i].owner,
 get_updates(place[i].update)) ;
}

    recInfo (globTid, "TRANSITIONS [%i]\n", num_tr) ;
    for (i=0; i<num_tr; i++)
        {
/* recInfo (globTid, "\t%6s  %4.1f  %4.1f  %2i  %2i\n", trans[i].name,
 trans[i].x_loc_in, trans[i].y_loc_in,
 cntArcs(trans[i].inp_arc), cntArcs(trans[i].out_arc)) ;*/
recInfo (globTid, "\t%12s  %8x  %4x  %s\n", trans[i].name,
 trans[i].id, trans[i].owner,
 get_updates(trans[i].update)) ;
}

    }


/*******************************************************************************
```

```
  cntArcs  - count number of arcs in list
*********************************************************************************/
int  cntArcs (struct Arc *p)
    {
    int  cnt = 0 ;

    while (p)
        {
p = p->next ;
cnt++ ;
}
    return (cnt) ;
    }


/*********************************************************************************
  register_remote  - sends request msgs to register remote nodes
*********************************************************************************/
STATUS  register_remote ()
    {
    int         i ;
    REG_REMOTE  *ps ;

    /*  send msg for each remote place  */
    for (i=0; i<num_pl; i++)
        {
if (place[i].owner != globTid)
    {
    ps = (REG_REMOTE *) malloc (sizeof (REG_REMOTE)) ;
    strcpy (ps->sym_name, place[i].name) ;
    strcpy (ps->act_name, place[i].act_name) ;
    ps->type = PLACE_NODE ;
    msgBuildSend (place[i].owner, globTid, MSG_REGISTER_REMOTE,
  (void *) ps, sizeof (REG_REMOTE),
  MF_STANDARD) ;
    }
        }

    /*  send msg for each remote transition  */
    for (i=0; i<num_tr; i++)
        {
if (trans[i].owner != globTid)
    {
    ps = (REG_REMOTE *) malloc (sizeof (REG_REMOTE)) ;
    strcpy (ps->sym_name, trans[i].name) ;
    strcpy (ps->act_name, trans[i].act_name) ;
    ps->type = TRANS_NODE ;
    msgBuildSend (trans[i].owner, globTid, MSG_REGISTER_REMOTE,
  (void *) ps, sizeof (REG_REMOTE),
  MF_STANDARD) ;
    }
        }

    return (OK) ;
    }


/*********************************************************************************
  confirm_remote  - send return msg with remote node's id
*********************************************************************************/
STATUS  confirm_remote (TID_TYPE reqTid, REG_REMOTE *ps)
```

```
    {
    int          indx ;
    REG_REMOTE   *pret ;

    /*  allocate return msg  */
    pret = (REG_REMOTE *) malloc (sizeof (REG_REMOTE)) ;
    strcpy (pret->sym_name, ps->sym_name) ;
    strcpy (pret->act_name, ps->act_name) ;
    pret->type = ps->type ;

    /*  find place or transition  */
    switch (ps->type)
        {
case PLACE_NODE:
     /*  find local index  */
     if ((indx = find_place ((char *)ps->act_name)) < 0)
        {
recInfo (globTid, "Could not find place '%s' by name\n",
                                        ps->act_name) ;
free (pret) ;
return (ERROR) ;
        }

     /*  send return msg containing id  */
     pret->id   = MAKE_ID (globTid, PLACE_NODE, indx) ;
     msgBuildSend (reqTid, globTid, MSG_CONFIRM_REMOTE,
   (void *) pret, sizeof (REG_REMOTE),
   MF_STANDARD) ;
     break ;

case TRANS_NODE:
     /*  find local index  */
     if ((indx = find_trans ((char *)ps->act_name)) < 0)
        {
recInfo (globTid, "Could not find trans '%s' by name\n",
                                    ps->act_name) ;
free (pret) ;
return (ERROR) ;
        }

     /*  send return msg containing id  */
     pret->id   = MAKE_ID (globTid, TRANS_NODE, indx) ;
     msgBuildSend (reqTid, globTid, MSG_CONFIRM_REMOTE,
   (void *) pret, sizeof (REG_REMOTE),
   MF_STANDARD) ;
     break ;
        }
    return (OK) ;
    }


/*****************************************************************************
  save_remote   - save data from confirm remote msg
*****************************************************************************/
STATUS  save_remote (REG_REMOTE *ps)
    {
    int  indx ;

    /*  find place or transition  */
    switch (ps->type)
```

```
         {
case PLACE_NODE:
      /*  find local index  */
      if ((indx = find_place ((char *)ps->sym_name)) < 0)
         {
recInfo(globTid,"Could not find place '%s' by name\n",
                                        ps->sym_name);
return (ERROR) ;
         }

      /*  save id  of remote place  */
      place[indx].id = ps->id ;
      break ;

case TRANS_NODE:
      /*  find local index  */
      if ((indx = find_trans ((char *)ps->sym_name)) < 0)
         {
recInfo(globTid,"Could not find trans '%s' by name\n",
                                         ps->sym_name);
return (ERROR) ;
         }

      /*  save id  of remote trans  */
      trans[indx].id = ps->id ;
      break ;
         }
     return (OK) ;
     }


/*****************************************************************************
  save_update  - save data from update register & reply with node id
*****************************************************************************/
STATUS  save_update (MSG_TYPE  *m)
     {
     int          indx ;
     int          id ;
     char        *name ;
     UPDATES     *pupd ;
     TID_TYPE     myTid ;
     REG_REMOTE  *ps ;

     /*  parse message  */
     myTid  = m->dest ;
     ps     = (REG_REMOTE *) m->data ;

     /*  find place or transition  */
     switch (ps->type)
         {
case PLACE_NODE:
      /*  find local index  */
      if ((indx = find_place ((char *)ps->act_name)) < 0)
         {
recInfo (myTid,"Could not find place '%s' by name\n",
                                             ps->act_name) ;
return (ERROR) ;
         }

      /*  save id for reply msg  */
      id = place[indx].id ;
```

```
        /*  add to update list  */
        pupd = (UPDATES *) malloc (sizeof (UPDATES)) ;
        pupd->next = place[indx].update ;
        place[indx].update = pupd ;
        pupd->tid = m->source ;
        pupd->num = ps->id ;
        break ;

    case TRANS_NODE:
        /*  find local index  */
        if ((indx = find_trans ((char *)ps->act_name)) < 0)
           {
recInfo (myTid,"Could not find trans '%s' by name\n",
                                                ps->act_name) ;
return (ERROR) ;
           }

        /*  save id for reply msg  */
        id = trans[indx].id ;

        /*  add to update list  */
        pupd = (UPDATES *) malloc (sizeof (UPDATES)) ;
        pupd->next = trans[indx].update ;
        trans[indx].update = pupd ;
        pupd->tid = m->source ;
        pupd->num = ps->id ;
                pupd->which = ps->which;
/*          recInfo(globTid,"Saved update for trans %s, update = %d\n",
                                   trans[indx].name, trans[indx].update); */
        break ;
    }

      /*  send id in reply msg  */
      msgReply (m, (void *) id, MS_NONE, MF_STANDARD) ;

      return (OK) ;
      }


/******************************************************************************
   initial_marking  - send initial marking to displays
******************************************************************************/
STATUS  initial_marking ()
      {
      int         i ;
      UPDATES   *pu ;

      /*  check each place  */
      for (i=0; i<num_pl; i++)
         {
/*  skip places with no marking, or no updates  */
if ((place[i].orig_token == 0) ||
      ((pu = place[i].update) == NULL))
      continue ;

/*  loop through list  */
while (pu)
      {
      msgBuildSend (pu->tid, globTid, MSG_SET_TOK,
   (void *) ((pu->num & 0x0000ffff)
  | (place[i].orig_token << 16)),
  MS_NONE, MF_STANDARD) ;

/*        recInfo (globTid,"Initial marking=%2i in place '%s.%s' on display %x\n",
     place[i].orig_token, myName, place[i].name, pu->tid) ; */

    pu = pu->next ;
    }
        }

    return (OK) ;
    }
```

C.3 *Xplay.c* — Tests and Fires Transitions

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

#include <stdio.h>

#include "ctos.h"
#include "btsLib.h"

#include "Global_def.h"
#include "msg.h"
#include "Global_var.p"
#include "Xplay.p"

extern char *myName ;

/**************************************************************************
*   get_ready_trans()
*      This routine selects the next transition to fire.  It is
*    currently based on a pseudo-random method, with some
*    consideration given to speed.  It returns -1 if no
*    transitions are enabled.
*/

int get_ready_trans(void)
{
 int current, start;

 start = random() % num_tr;
 current = start;
 do
    {
/* recInfo(0,"%s: Checking transition %i out of %i\n",
                               myName, current, num_tr) ;*/
     if (trans_tok_enabled(current))
        {if (trans_pre_enabled(current))
            return(current);
        };
     current = (current+1)%num_tr;
    }
    while(current != start);
 return(-1);
}
```

```
/*****************************************************************************
*   get_ready_hidden_trans()
*      This routine selects the next hidden transition to fire.  It is
*    currently based on a pseudo-random method, with some
*    consideration given to speed.  It returns -1 if no
*    transitions are enabled.
*/

int get_ready_hidden_trans(void)
{
 int current, start;

 start = random() % num_tr;
 current = start;
 do
    {
/* recInfo(0,"%s: Checking transition %i out of %i\n",
                            myName, current, num_tr) ;*/
    if (hidden(current) == TRUE)
       {if (trans_tok_enabled(current))
          {if (trans_pre_enabled(current))
              return(current);
          };
       };
     current = (current+1)%num_tr;
    }
    while(current != start);
 return(-1);
}

/*****************************************************************************
*   hidden()
*      This routine checks if the transition is not visible
*    on any display.
*/

int hidden(int t_num)
{
 UPDATES *p;

 p = trans[t_num].update;
 while (p != NULL)
    {if ((p->which == T_START)||(p->which == T_BOTH))
        {/* recInfo(globTid,"Trans %s is visible\n",trans[t_num].name); */
         return(FALSE);
        };
     p = p->next;
    };

/* recInfo(globTid,"Trans %s is hiddon with update: %d\n",
                          trans[t_num].name, trans[t_num].update); */
 return(TRUE);
}

/*****************************************************************************
*   done_firing()
*      This routine checks if the fired transition should
*    complete the firing cycle immediately.  Only blocking
*    transitions should not finish immediately.
*/
```

```
int done_firing(int t_num)
{
 if (trans[t_num].post_type == F_REMOTE)
    return(FALSE);
 else
    return(TRUE);
}

/*************************************************************************
*   set_token()
*      This routine sets the number of tokens in the given place.
*/

void set_token(int data)
{
 int p_num, num_tok;

 num_tok = (data&0xffff0000)>>16;
 p_num = data&0xffff;

 place[p_num].curr_token = num_tok;

 disp_set_tok(p_num);
}

/*************************************************************************
*   sub_token()
*      This routine subtracts a number of tokens to the given place.
*/

void sub_token(int data)
{
 int p_num, num_tok;

 num_tok = (data&0xffff0000)>>16;
 p_num = data&0xffff;

 place[p_num].curr_token -= num_tok;
 if (place[p_num].curr_token < 0)
    place[p_num].curr_token = 0;

 disp_set_tok(p_num);
}

/*************************************************************************
*   add_token()
*      This routine adds a number of tokens to the given place.
*/

void add_token(int data)
{
 int p_num, num_tok;

 num_tok = (data&0xffff0000)>>16;
 p_num = data&0xffff;

 place[p_num].curr_token += num_tok;
 if (place[p_num].curr_token < 0)
    place[p_num].curr_token = 0;

 disp_set_tok(p_num);
```

```
}

/***************************************************************************
*   trans_pre_enabled()
*      This routine checks if both the tape requirements and
*    the enabling function requirements for firing a
*    transition are met, as well as if the transition is
*    currently not firing.
*/

int trans_pre_enabled(int i)
{
 int enabled = TRUE;

 if (trans[i].firing)
    return(FALSE);

 if (trans[i].tape_req)
    {if (trans[i].tape_req != query_tape())
        return(FALSE);
    };

 if (trans[i].enable_func != NULL)
    {if (trans[i].enable_type == F_LOCAL)
        enabled = (*trans[i].enable_func)(trans[i].enable_action.
                                           trans[i].enable_va
    };

/* recInfo(globTid,"Transition %d enabling: %d\n",i,enabled); */
 return(enabled);
}

/***************************************************************************
*   trans_tok_enabled()
*      This routine checks if the token requirements for firing
*    a transition are met.  Both multiplicity and inhibitor arcs
*    are checked.
*/

int trans_tok_enabled(int i)
{
 struct Arc *arc_a;

 arc_a = trans[i].inp_arc;
 while(arc_a)
    {if (place[arc_a->place].curr_token < arc_a->mult)
        return(FALSE);
     arc_a = arc_a->next;
    };

 arc_a = trans[i].inh_arc;
 while(arc_a)
    {if (place[arc_a->place].curr_token)
       return(FALSE);
     arc_a = arc_a->next;
    };

 return(TRUE);
}

/***************************************************************************
```

```
*   fire_start()
*      This routine starts the firing of a transition.  It removes
*    tho appropriate tokens, absorbs any tape command and calls
*    the post_function.
*/

void fire_start(int i)
{struct Arc *arc_a;

 trans[i].firing = TRUE;

 arc_a = trans[i].inp_arc;
 while(arc_a)
    {place[arc_a->place].curr_token -= arc_a->mult;
     disp_set_tok(arc_a->place);
     arc_a = arc_a->next;
    };

 if (trans[i].tape_req)
    {absorb_tape();
    };

 if (trans[i].post_func)
    if (trans[i].post_type == F_LOCAL)
        (*trans[i].post_func)(trans[i].post_action, trans[i].post_var);
    else
        {(*trans[i].post_func)(i, trans[i].post_tid,
                             trans[i].post_action, trans[i].post_data);
         recInfo(globTid,"%s: Executed remote post function\n",myName);
        };

 disp_start_fire(i);
}

/***********************************************************************
*   fire_end()
*      This routine ends the firing of a transition.  It updates
*    the number of tokens in each output place.
*/

void fire_end(int i)
{
 struct Arc *arc_a;

 trans[i].firing = FALSE;

 arc_a = trans[i].out_arc;
 while(arc_a)
    {place[arc_a->place].curr_token += arc_a->mult;
     disp_set_tok(arc_a->place);
     arc_a = arc_a->next;
    };

 disp_end_fire(i);
}
```

C.4 *enable_func.c* — Local and Remote Enabling Functions

```
#include <stdio.h>
```

```
#include "ctos.h"
#include "domLib.h"
#include "Global_def.h"
#include "Global_var.p"

/*************************************************************************
*   Assorted enabling_functions()
*      Below is a bunch of small enabling functions.  Done and
*    not_done and used for looping
*/

int gt(int action, int data)
{
 int var;
 char buff[20];

 recPrompt(globTid, buff, "Enter greater than %d to fire: ", data);
 sscanf(buff,"%d",&var);
 if (data < var)
    return(TRUE);
  else
    return(FALSE);
}

int lt(int action, int data)
{
 int var;
 char buff[20];

 recPrompt(globTid,buff,"Enter less than %d to fire: ", data);
 sscanf(buff,"%d",&var);
 if (data > var)
    return(TRUE);
  else
    return(FALSE);
}

int done(int action, int data)
{
 if (action == 1)          /* Return TRUE if variable = 0 */
    {if (read_var(data))
       return(FALSE);
     else
       return(TRUE);
    }
 else if (action == -1)    /* Return TRUE if variable != 0 */
    {if (read_var(data))
       return(TRUE);
     else
       return(FALSE);
    }
}

int not_done(int action, int data)
{
 if (!read_var(data))
    return(FALSE);
  else
    return(TRUE);
}
```

```
int DOM_bool(int action, int data)
{
 int *DOM_value;
 char *str_ptr;

 str_ptr = get_var_name(data);
recInfo(globTid,"Executing DOM_bool for %s.  I hope you wanted to!\n",str_ptr);

 if ((DOM_value = (int *)objCopy(globTid,str_ptr)) == NULL)
    {recInfo(globTid,"ERROR: Object %s not found in Data Object Manager\n",
                                                               str_ptr);
     return(FALSE);
    };

 if (action == 1)
    {if (*DOM_value)
       return(TRUE);
     else
       return(FALSE);
    }
 else if (action == -1)
    {if (*DOM_value)
       return(FALSE);
     else
       return(TRUE);
    }
}

int DOM_done(int action, int data)
{
 int *DOM_value;

 if ((DOM_value = (int *)objCopy(globTid,data)) == NULL)
    {recInfo(globTid,"ERROR: Object %s not found in Data Object Manager\n",
                                                                data);
     return(FALSE);
    };

 if (action == 1)
    {if (*DOM_value)
       retu-n(FALSE);
     else
       return(TRUE);
    }
 else if (action == -1)
    {if (*DOM_value)
       return(TRUE);
     else
       return(FALSE);
    }
}

/*************************************************************************
*   enable_table[]
*      This array equates the symbolic names of the enabling
*    functions with the actual function pointer.
*/

#define MAX_ENABLE 6

struct func_entry enable_table[] =
   {{DOM_bool,"DOM_bool"},
    {DOM_done,"DOM_done"},
    {gt,"greater_than"},
    {lt,"less_than"},
    {done,"done"},
    {not_done,"not_done"}
   };

/*************************************************************************
*    find_enable()
*       This routine returns a pointer to the enabling function
*     specified by its symbolic name.
*/

void *find_enable(name)
   char *name;
{
 int i;

 if (!strcmp(name,"0"))
    return(NULL);

 for(i=0;i<MAX_ENABLE;i++)
    {if (!strcmp(name,enable_table[i].ident_str))
       return((void *)enable_table[i].func_ptr);
    };

 return((void *)-1);
}
```

C.5 *post_func.c* — Local and Remote Post Functions

```
#include <stdio.h>
#include "Global_def.h"
#include "Global_var.p"
#include "domLib.h"
#include "coordTools.h"
#include "msg.h"

/*************************************************************************
*    Assorted post_functions()
*       Below is a bunch of small post functions.  Initialize and
*     decrement are used for looping, but only have one
*     variable to work with!  Do not use two loops with these....
*/

int alive(int action, int data)
{
 recInfo(globTid,"Alive: %d\n", action);
}

int live_init(int action, int data)
{
 char buff[10];
 int  val;

/* recInfo(globTid,"Inside initialize\n"); */
 recPrompt(globTid, buff, "Enter variable value: ");
 sscanf(buff,"%d",&val);
 save_var(data, val);
}

int initialize(int action, int data)
{
 save_var(data, action);
}

int decrement(int action, int data)
{
 int val;
 val = read_var(data);
 save_var(data, (val-action));
}

int increment(int action, int data)
{
 int val;
 val = read_var(data);
 save_var(data, (val+action));
}

int renameOBJ(int action, int data)
{
 char *str_ptr;

 str_ptr = (char *)get_var_name(data);
 recInfo(globTid,"Obj before rename= '%s'\n", str_ptr) ;
 if (renameObjs(globTid, str_ptr) == ERROR)
    recInfo(globTid,"renameObjs Error: could not rename '%s'\n",str_ptr);
 else
    recInfo(globTid,"renameObjs successfully renamed '%s'\n",str_ptr);

 str_ptr = (char *)get_var_name(data);
 recInfo(globTid,"Obj after rename= '%s'\n", str_ptr) ;
}

int duplicateOBJ(int action, int data)
{
 char *str_ptr;

 str_ptr = (char *)get_var_name(data);
 recInfo(globTid,"Obj before dup= '%s'\n", str_ptr) ;
 if (duplicateObjs(globTid, str_ptr) == ERROR)
    recInfo(globTid,"dupObjs Error: could not dup '%s'\n",str_ptr);
 else
    recInfo(globTid,"dupObjs successfully dup'd '%s'\n",str_ptr);

 str_ptr = (char *)get_var_name(data);
 recInfo(globTid,"Obj after dup= '%s'\n", str_ptr) ;
}

/*************************************************************************
*   send_post_msg()
*      This routine builds and sends a message to the appropriate
*    post function server.
*/
int  send_post_msg(int indx, TID_TYPE dest_tid, int action, char *data)
{

 POST_DATA p_data;
 p_data.indx = indx;
 p_data.action = action;
 strcpy(p_data.data_obj, data);

 msgBuildSend(dest_tid, globTid, MSG_TRANS_FIRED, (void *)&p_data,
                          sizeof(p_data), MF_STANDARD);
}


/**************************************************************************
*   post_table[]
*      This array equates the symbolic names of the post function
*    with the actual function pointer.
*/

#define MAX_POST 8

struct func_entry post_table[] =
   {{send_post_msg,REMOTE_POST_FUNC},
    {renameOBJ,"renameObjs"},
    {duplicateOBJ,"dupObjs"},
    {alive,"alive"},
    {initialize,"initialize"},
    {live_init,"live_init"},
    {decrement,"decrement"},
    {increment,"increment"}
   };

/**************************************************************************
*   find_post()
*      This routine returns a pointer to the post function
*    specified by its symbolic name.
*/

void *find_post(name)
   char *name;
{
 int i;

 if (!strcmp(name,"0"))
    return(NULL);

 for(i=0;i<MAX_POST;i++)
    {if (!strcmp(name,post_table[i].ident_str))
       return((void *)post_table[i].func_ptr);
    };

 return((void *)-1);
}

/*                                */
/*  "rename" POST_LOCAL function  */
/*                                */
STATUS  renameObjs (TID_TYPE t, char *obj_string)
    {
#   define  MAX_OBJS  16
    int     i, num_objs ;
    char    *name[MAX_OBJS] = {NULL} ;
    char    *objs ;
    objs = (char *) malloc (strlen(obj_string)+1) ;
    strcpy (objs, obj_string) ;

    if ((num_objs = parseNames (objs, name)) == ERROR)
        {
recInfo(t,"renameObjs ERROR: could not parse '%s'\n", obj_string) ;
free (objs) ;
return (ERROR) ;
        }

    for (i=0; i<num_objs; i+=2)
        {
if ((name[i]==NULL)||(name[i+1]==NULL))
    {
    recInfo(t,"renameObjs ERROR: one of the names was null\n") ;
    free (objs) ;
    return (OK) ;
    }

if (objRename(t, name[i], name[i+1]) == ERROR)
    {
    free (objs) ;
    recInfo(t,"renameObjs ERROR: objRename returned error\n") ;
    return (ERROR) ;
    }
recInfo(t,"Successfully renamed '%s' as '%s'\n", name[i], name[i+1]) ;
}

    free (objs) ;
    return (OK) ;
    }


/*                                      */
/*  "duplicate" POST_LOCAL function     */
/*                                      */
STATUS  duplicateObjs (TID_TYPE t, char *obj_string)
    {
#   define  MAX_OBJS  16
    int      i, num_objs ;
    char    *name[MAX_OBJS] = {NULL} ;
    char    *objs ;

    objs = (char *) malloc (strlen(obj_string)+1) ;
    strcpy (objs, obj_string) ;

    if ((num_objs = parseNames (objs, name)) == ERROR)
        {
recInfo(t,"duplicateObjs ERROR: could not parse '%s'\n", obj_string) ;
free (objs) ;
return (ERROR) ;
        }

    for (i=0; i<num_objs; i+=2)
        {
if ((name[i]==NULL)||(name[i+1]==NULL))
    {
    recInfo(t,"duplicateObjs ERROR: one of the names was null\n") ;
    free (objs) ;
    return (OK) ;
    }
if (objDup(t, name[i], name[i+1]) == ERROR)
    {
    recInfo(t;"duplicateObjs ERROR: objDup returned error\n") ;
    free (objs) ;
    return (ERROR) ;
    }
recInfo(t,"Successfully duplicated '%s' as '%s'\n",
name[i], name[i+1]) ;
}

    free (objs) ;
    return (OK) ;
    }
```

C.6 *tape.c* — Tape Commands

```
#include <stdio.h>

#include "Global_def.h"
#include "Global_var.p"

/***********************************************************************
*   init_tape()
*      This routine initializes the tape list.  You should use
*     clear_tape() instead, as it free's the memory used by
*     the tape elements as well.
*/

void init_tape(void)
{
 head = NULL;
 tail = NULL;
}

/***********************************************************************
*   append_tape()
*      This routine appends a tape command onto the end of the
*     tape list.  This is the standard method of adding a tape
*     command.
*/

void append_tape(int command)
{
 struct tape_node *tmp;

 tmp = (struct tape_node *)malloc(sizeof(struct tape_node));

 if (head)
    {tail->next = tmp;
     tail = tmp;
     tail->id = command;
     tail->next = NULL;
    }
  else
    {head = tmp;
     tail = head;
     tail->id = command;
     tail->next = NULL;
    };
}

/*************************************************************************
*   insert_tape()
*      This routine inserts a tape command onto the head of the
*    tape list.  This is not a standard use of the tape, but
*    provides for greater flexibility and exception handling.
*/

void insert_tape(int command)
{
 struct tape_node *tmp;

 tmp = (struct tape_node *)malloc(sizeof(struct tape_node));

 if (head)
    {tmp->next = head;
     head = tmp;
     head->id = command;
    }
  else
    {head = tmp;
     tail = head;
     head->id = command;
    };
}

/*************************************************************************
*   query_tape()
*      This routine returns the command at the head of the tape,
*    without removing it.  This is used to check if a transition's
*    tape requirement is satisfied for enabling.
*/

int query_tape(void)
{
 if (head)
    return(head->id);
  else
    return(-1);
}

/*************************************************************************
*   absorb_tape()
*      This routine returns the command at the head of the tape,
*    then removes it from the list.
*/

int absorb_tape(void)
{
 struct tape_node *tmp;
 int id;

 if (head)
    {id = head->id;
     tmp = head;
     if (head == tail)
        tail = NULL;
     head = head->next;
     free(tmp);
    }
  else
    id = -1;

return(id);
}

/*************************************************************************
*   clear_tape()
*      This routine removes all of the tape commands from the
*    the tape list, and sets the head and tail pointers to NULL.
*/

void clear_tape(void)
{
 struct tape_node *tmp1,*tmp2;

 tmp1 = head;

 while(tmp1)
    {tmp2 = tmp1->next;
     free(tmp1);
     tmp1 = tmp2;
    };

 head = NULL;
 tail = NULL;
}

/*************************************************************************
*   find_tape()
*      This routine returns the tape command id for the given
*    symbolic name.  If the name is not valid, -1 is returned.
*/

int find_tape(char *name)
{
 int i;

 if (!strcmp(name,"0"))
    return(NULL);

 for(i=0;i<MAX_TAPE;i++)
    {if (!strcmp(name,tape_table[i].ident_str))
        return(tape_table[i].id);
    };

 return(-1);
}

/*************************************************************************
*   find_tape_name()
*      This routine returns the symbolic name of the given
*    tape command id.  If the id is not valid, NULL is returned.
*/

char *find_tape_name(int id)
{
 int i;

 for(i=0;i<MAX_TAPE;i++)
```

```
    {if (id == tape_table[i].id)
        return(tape_table[i].ident_str);
    };

 return(NULL);
}

/*************************************************************************
*   get_tape_command()
*      This routine queries the user for a tape command, and
*    returns his selection.  This is used for testing purposes.
*/

int get_tape_command(void)
{
 int select, i;

 fprintf(stderr,"\nTape Commands:\n");
 for(i=0;i<MAX_TAPE;i++)
    fprintf(stderr," %d: %s\n",i,tape_table[i].ident_str);
 fprintf(stderr," %d: No selection\n",i);

 fprintf(stderr,"\nSelection: ");

 fscanf(stdin,"%d",&select);

 if (select == MAX_TAPE)
    {fprintf(stderr,"No selection\n");
     return(0);
    };

 if ((select<0)||(select>MAX_TAPE))
    {fprintf(stderr,"%d is not a valid command\n");
     return(0);
    };

 fprintf(stderr,"%s\n",tape_table[select].ident_str);

 return(tape_table[select].id);
}
```

C.7 *load_net.c* — Loads the Petri Net Configuration File

```
/****************************************************************************
    File:     load_net.c

    Purpose:  reads player and display configuration files

    Author:   Joe Peck
              revised:  Don Lefebvre

    Revised:  20 Aug'91
****************************************************************************/
/*------------------------------------ include files --------------------*/
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <stdio.h>
#include <math.h>
```

```
#include "Global_def.h"
#include "Global_var.p"
#include "recLib.h"
#include <ctos.h>

extern char *myName ;

/*------------------------------------ function prototypes -------------*/
void     *find_enable        () ;
void     *find_post          () ;
int       find_place         (char *name) ;
int       find_trans         (char *name) ;
void      do_error           (char *s) ;
STATUS    read_subnet_config (FILE *infile) ;
STATUS    read_display_config (FILE *infile) ;
TID_TYPE  find_owner         (char *owner) ;


/****************************************************************************
  load_net  - load configuration file
-----------------------------------------------------------------------------
  RETURNS:  OK or ERROR indicating success in reading config file

  INPUTS:   char *net_config_file  - name of configuration file
            int  type              - type of config file (CF_PNET or CF_DISP)
****************************************************************************/
STATUS  load_net (char *net_config_file, int type)
{
 FILE   *fp;                  /* Temporary file pointer          */
 char   file_name[MAXLINE];   /* Actual file to open             */
 int    i, j, k, l;           /* Generic loop variables          */
 char   linebuf[MAXLINE];     /* Generic buffer for line reads   */
 char   name[MAXLINE];        /* Used to read object names       */
 FILE   *infile;              /* File pointer to input file      */
 char   comment[10][MAXLINE]; /* Allow for ten lines of comments */
 struct Arc *arc_a, *arc_b;   /* Pointers to arc structures      */
 struct Arc_point *arc_point_a, *arc_point_b;
 int    num_points;           /* Number of points in an arc      */
 int    num_arcs;             /* Number of arcs per type         */
 int    dummy;                /* Generic variable                */
 int num_read;                /* Number of items read by fscanf  */


   /*  Create config file name  */

   switch (type)
       {
       case CF_DISP:
           sprintf (file_name, "%s.%s", net_config_file, "disp") ;
   break ;

       case CF_PNET:
   sprintf (file_name, "%s.%s", net_config_file, "pnet") ;
   break ;

       default:
   strcpy (file_name, net_config_file) ;
       }

   /*  Open the config file  */
```

```
if (fp = fopen(file_name,"r"))
    {
    infile = fp;
    }
else
    {
    sprintf (linebuf, "Could not open config file %s", file_name) ;
    do_error (linebuf) ;
    return (ERROR) ;
    }

/* Interpret Comment */

while((int)fgets(linebuf,MAXLINE,infile)!=EOF)
   {if ((linebuf[0] == '|')&&(linebuf[1] == 'O')&&(linebuf[2] == '|'))
       break;
   };

i = 0;
while((int)fgets(comment[i],MAXLINE,infile)!=EOF)
   {if ((comment[i][0] == '|')&&(comment[i][1] == '\n'))
       {comment[i][0] = NULL;
        break;
       };
    if (++i > 9)
       i = 9;
   };

/* Get the number of objects in the net */

fscanf(infile,"f %d %d %d %d %d %d\n", &num_mp, &num_pl, &num_rp,
                  &num_tr, &num_gr, &num_cs);

/* Allocate work arrays */

place = (struct Place *)malloc(num_pl*sizeof(struct Place));
trans = (struct Trans *)malloc(num_tr*sizeof(struct Trans));
if (num_mp > 0)
   mpar  = (struct MPar *)malloc(num_mp*sizeof(struct MPar));
if (num_rp >0)
   rpar  = (struct RPar *)malloc(num_rp*sizeof(struct RPar));

/* Process marking parameters */

for (i=0;i<num_mp;i++)
   {num_read = fscanf(infile,"%s %d %f %f %d", name, &mpar[i].num_token,
                &mpar[i].tag_x_in, &mpar[i].tag_y_in, &mpar[i].layer);

    if (num_read != 5)
       do_error("Missing information for marking parameter");

    dummy = mpar[i].layer;   /* Each layer is indicated by a bit mask */
    mpar[i].layer = 1;
    while (dummy)
       {mpar[i].layer |= 1<<dummy;
        fscanf(infile,"%d",&dummy);
       };

    mpar[i].name = (char *)malloc(strlen(name)+1);
    strcpy(mpar[i].name,name);
   };

/* Process place information */

for (i=0;i<num_pl;i++)
   {num_read = fscanf(infile,"%s %d %f %f %f %f %d", name,
          &place[i].orig_token, &place[i].x_loc_in, &place[i].y_loc_in,
          &place[i].tag_x_in, &place[i].tag_y_in, &place[i].layer);

    if (num_read != 7)
       do_error("Missing information for place definition");

    dummy = place[i].layer;
    place[i].layer = 1;
    while (dummy)
       {place[i].layer |= 1<<dummy;
        fscanf(infile,"%d",&dummy);
       };

    place[i].name = (char *)malloc(strlen(name)+1);
    strcpy(place[i].name,name);

    place[i].curr_token = place[i].orig_token;
    if (place[i].orig_token < 0)          /* decode place marking */
       {place[i].curr_token = mpar[-place[i].orig_token - 1].num_token;
        place[i].orig_token = place[i].curr_token;
       };

   };

/* Process rate parameters */

for (i=0;i<num_rp;i++)
   {num_read = fscanf(infile,"%s %f %f %f %d", name, &rpar[i].fire_rate,
               &rpar[i].tag_x_in, &rpar[i].tag_y_in, &rpar[i].layer);

    if (num_read != 5)
       do_error("Missing information for rate parameter");

    dummy = rpar[i].layer;
    rpar[i].layer = 1;
    while (dummy)
       {rpar[i].layer |= 1<<dummy;
        fscanf(infile,"%d",&dummy);
       };

    rpar[i].name = (char *)malloc(strlen(name)+1);
    strcpy(rpar[i].name,name);
   };

/* Ignore groups currently */

if (num_gr)      /* get rid of last \n after previous line */
   fscanf(infile,"\n");
for (i=0;i<num_gr;i++)
   fgets(linebuf,MAXLINE,infile);

/* Process Transition Information */

for (i=0;i<num_tr;i++)
   {num_read =fscanf(infile,"%s %f %d %d %d %d %f %f %f %f %f %f %d", name,
            &trans[i].fire_rate, &dummy, &trans[i].kind,
            &num_arcs, &trans[i].orient, &trans[i].x_loc_in,
            &trans[i].y_loc_in, &trans[i].tag_x_in, &trans[i].tag_y_in,
            &trans[i].rate_x_in, &trans[i].rate_y_in, &trans[i].layer);

   if (num_read != 13)
      do_error("Missing information for transition definition");

   dummy = trans[i].layer;
   trans[i].layer = 1;
   while (dummy)
      {trans[i].layer |= 1<<dummy;
       fscanf(infile,"%d",&dummy);
      };

   trans[i].name = (char *)malloc(strlen(name)+1);
   strcpy(trans[i].name,name);

  /* process input arc information */

  for (j=0;j<num_arcs;j++)
     {arc_a = (struct Arc *)malloc(sizeof(struct Arc));
      num_read = fscanf(infile,"%d %d %d %d", &arc_a->mult,&arc_a->place,
                                        &num_points,&arc_a->layer);

      if (num_read != 4)
         do_error("Missing information for input arc definition");

      dummy = arc_a->layer;
      arc_a->layer = 1;
      while (dummy)
         {arc_a->layer |= 1<<dummy;
          fscanf(infile,"%d",&dummy);
         };

      arc_a->type = INPARC;
      arc_a->place -= 1;     /* GreatSPN arrays start at 1 ... yuck */

      if (j==0)
         {trans[i].inp_arc = arc_a;
          arc_b = arc_a;
         }
       else
         {arc_b->next - arc_a;
          arc_b = arc_a;
         };
      for (k=0;k<num_points ;k++)
         {arc_point_a = (struct Arc_point *)malloc(sizeof
                                             (struct Arc_point));
          num_read = fscanf(infile,"%f %f\n", &arc_point_a->x_in,
                                                &arc_point_a->y_in);
          if (num_read != 2)
             do_error("Missing input arc location information");

          if (k==0)
             {arc_b->arc_points = arc_point_a;
              arc_point_b = arc_point_a;
             }
           else
             {arc_point_b->next = arc_point_a;
              arc_point_b = arc_point_a;
              };

          };
       arc_a->next = NULL;
      };

  /* process output arc information */

  num_read = fscanf(infile,"%d\n",&l);

  if (num_read != 1)
     do_error("Missing number of output arcs");

  for (j=0;j<l;j++)
     {arc_a = (struct Arc *)malloc(sizeof(struct Arc));
      num_read = fscanf(infile,"%d %d %d %d", &arc_a->mult,&arc_a->place,
                                 &num_points,&arc_a->layer);

      if (num_read != 4)
         do_error("Missing information for output arc definition");

      dummy = arc_a->layer;
      arc_a->layer = 1;
      while (dummy)
         {arc_a->layer |= 1<<dummy;
          fscanf(infile,"%d",&dummy);
         };

      arc_a->type = OUTARC;
      arc_a->place -= 1;

      if (j==0)
         {trans[i].out_arc = arc_a;
          arc_b = arc_a;
         }
       else
         {arc_b->next = arc_a;
          arc_b = arc_a;
         };
      for (k=0;k<num_points ;k++)
         {arc_point_a = (struct Arc_point *)malloc(sizeof
                                         (struct Arc_point));
          num_read = fscanf(infile,"%f %f\n", &arc_point_a->x_in,
                                            &arc_point_a->y_in);
          if (num_read != 2)
             do_error("Missing output arc location information");

          if (k==0)
             {arc_b->arc_points = arc_point_a;
              arc_point_b = arc_point_a;
             }
           else
             {arc_point_b->next = arc_point_a;
              arc_point_b = arc_point_a,
             };

         };
     };

  /* process inhibitor arc information */
```

```
      num_read = fscanf(infile,"%d\n",&l);

      if (num_read != 1)
         do_error("Missing number of inhibitor arcs");

      for (j=0;j<l;j++)
         {arc_a = (struct Arc *)malloc(sizeof(struct Arc));
          num_read = fscanf(infile,"%d %d %d %d", &arc_a->mult,&arc_a->place,
                                                &num_points,&arc_a->layer);

          if (num_read != 4)
             do_error("Missing information for inhibitor arc definition");

          dummy = arc_a->layer;
          arc_a->layer = 1;
          while (dummy)
             {arc_a->layer |= 1<<dummy;
              fscanf(infile,"%d",&dummy);
             };

          arc_a->type = INHARC;
          arc_a->place -= 1;

          if (j==0)
             {trans[i].inh_arc = arc_a;
              arc_b = arc_a;
             }
           else
             {arc_b->next = arc_a;
              arc_b = arc_a;
             };
          for (k=0;k<num_points ;k++)
             {arc_point_a = (struct Arc_point *)malloc(sizeof
                                                (struct Arc_point));
              num_read = fscanf(infile,"%f %f\n", &arc_point_a->x_in,
                                                     &arc_point_a->y_in);
              if (num_read != 2)
                 do_error("Missing inhibitor arc location information");

              if (k==0)
                 {arc_b->arc_points = arc_point_a;
                  arc_point_b = arc_point_a;
                 }
               else
                 {arc_point_b->next = arc_point_a;
                  arc_point_b = arc_point_a;
                 };

             };
         };
    };

  /*  now read subnet interconnection info  */
  switch (type)
      {
      case CF_DISP:
  if (read_display_config (infile) == ERROR)
      {
      close (infile) ;
      return (ERROR) ;
      }
```

```
   break ;

       case CF_PNET:
   if (read_subnet_config (infile) == ERROR)
       {
       close (infile) ;
       return (ERROR) ;
       }
   break ;
       }

   /*  wrapup and exit  */
   close (infile) ;
   return (OK) ;
}

/******************************************************************************
  find_place  - find place number corresponding to name
******************************************************************************/
int  find_place (char *name)
{
 int i;

 for(i=0;i<num_pl;i++)
    if (!strcmp(name,place[i].name))
       return(i);

 return(-1);
}

/******************************************************************************
  find_trans  - find transition number corresponding to name
******************************************************************************/
int  find_trans (char *name)
{
 int i;

 for(i=0;i<num_tr;i++)
    if (!strcmp(name,trans[i].name))
       return(i);

 return(-1);
}


/******************************************************************************
  do_error  - print error string ;
******************************************************************************/
void  do_error (char *s)
{
 recInfo(globTid,"ERROR: %s\n",s);
}

/******************************************************************************
  read_subnet_config  - read configuration of subnet player
******************************************************************************/
STATUS  read_subnet_config (FILE *infile)
   {char  linebuf    [255] ;
    char  ick        [15][255];
    char  sym_name   [40] ;
    char  buff       [40] ;
    char  func_name  [40] ;
    char  loc        [40] ;
    char  owner      [40] ;
    char  name       [40] ;
    char  action     [40] ;
    char  data       [255] ;
    int   indx ;
    int   i, imax;
    void  *ptr;

    /*  read remaining lines of file  */
    while (fgets (linebuf, MAXLINE, infile) != NULL)
        {
         imax = sscanf(linebuf, "%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s",
                  ick[0],ick[1],ick[2],ick[3],ick[4],ick[5],ick[6],ick[7],
                  ick[8],ick[9],ick[10],ick[11],ick[12],ick[13],ick[14]);
         i=1;

/*  process PLACE line  */

if (! strncasecmp (ick[0], "PLACE", 5))
    {
            strcpy(sym_name,ick[i++]);
            strcpy(loc,ick[i++]);

    if (! strncasecmp (loc, "REMOTE", 5))
        {
 if ((indx = find_place (sym_name)) < 0)
    {
    recInfo (globTid, "Could not find place '%s' by name\n",
     sym_name) ;
    continue ;
    }
                 place[indx].act_name = (char *)malloc(strlen(ick[i])+1);
                 strcpy(place[indx].act_name,ick[i++]);
                 strcpy(owner,ick[i++]);
        place[indx].owner = find_owner (owner) ;
        }
    else
        {
if ((indx = find_place (sym_name)) < 0)
    {
    recInfo (globTid, "Could not find place '%s' by name\n",
        sym_name) ;
    continue ;
    }
        place[indx].owner = globTid ;
        place[indx].id    = MAKE_ID (globTid, PLACE_NODE, indx) ;
                 place[indx].act_name = (char *)malloc(strlen(sym_name)+1);
                 strcpy(place[indx].act_name, sym_name);
        }
    }

/*  process TRANS line  */

else if (! strncasecmp (ick[0], "TRANS", 5))
           {strcpy(sym_name,ick[i++]);

    if ((indx = find_trans (sym_name)) < 0)
       {
        recInfo (globTid, "Could not find trans '%s' by name\n",
                                       sym_name) ;
                continue ;
        }
      trans[indx].owner = globTid ;
      trans[indx].id    = MAKE_ID (globTid, TRANS_NODE, indx) ;
           trans[indx].enable_type = F_NONE;
           trans[indx].post_type = F_NONE;
           trans[indx].update = NULL;

           while (i<imax)
              {strcpy(buff,ick[i++]);

                if (!strncasecmp(buff, "ENABLE_LOCAL", 12))
                   {
                    trans[indx].enable_type = F_LOCAL;
                    strcpy(func_name,ick[i++]);
                    ptr = find_enable(func_name);
                    if ((int)ptr == -1)
                       {recInfo (globTid, "Could not find enabling ",
                                             "function %s\n", func_name);
                         continue;
                       };
recInfo(globTid,"Found enabling function %s\n",func_name);
                    trans[indx].enable_func = ptr;

                    strcpy(action,ick[i++]);
                    trans[indx].enable_action = atoi(action);
                    strcpy(data,ick[i++]);
                    if ((trans[indx].enable_var = find_variable(data)) < 0)
                      recInfo(globTid, "Insufficient storage for '%s'\n",
                                                               data);
                   }
                else if (! strncasecmp(buff, "POST_LOCAL", 10))
                   {
                    trans[indx].post_type = F_LOCAL;
                    strcpy(func_name,ick[i++]);
                    ptr = find_post(func_name);
                    if ((int)ptr == -1)
                       {recInfo (globTid, "Could not find post function %s\n",
                                              func_name);
                         continue;
                       };
                    trans[indx].post_func = ptr;

                    strcpy(action,ick[i++]);
                    trans[indx].post_action = atoi(action);
                    strcpy(data,ick[i++]);
                    if ((trans[indx].post_var = find_variable(data)) < 0)
                      recInfo(globTid, "Insufficient storage for '%s'\n",
                                                    data);
                   }
                else if (! strncasecmp(buff, "POST_REMOTE", 11))
                   {
                    trans[indx].post_type = F_REMOTE;
                    trans[indx].post_func = find_post(REMOTE_POST_FUNC);
                    if ((int)ptr == -1)
                       {recInfo (globTid, "Could not find post function %s\n",
                                              REMOTE_POST_FUNC);
                         continue;
                       };
                    strcpy(func_name,ick[i++]);
                    trans[indx].post_tid = find_owner(func_name);

                    strcpy(action,ick[i++]);
                    trans[indx].post_action = atoi(action);
                    strcpy(data,ick[i++]);
                    trans[indx].post_data = (char *)malloc(strlen(data)+1);
                    strcpy(trans[indx].post_data, data);
                   };

        }

    }

/*  process SUBNET line  */

else if (! strncasecmp (ick[0], "SUBNET", 5))
    {
             strcpy(name,ick[i++]);
     net_name = (char *) malloc (strlen(name)+1) ;
     strcpy (net_name, name) ;
    }

/*  ignore all other lines  */
        else recInfo(globTid,"Unrecognized parameter in config file: %s\n",
                                        buff);

}
    return (OK) ;
    }

/******************************************************************************
  read_display_config  - read configuration of subnet display
******************************************************************************/
STATUS  read_display_config (FILE *infile)
    {
    char      linebuf [MAXLINE] ;
    char      ick      [15][40];
    char      buff     [40] ;
    char      sym_name [40] ;
    char      act_name [40] ;
    char      disp     [40] ;
    char      owner    [40] ;
    int       indx ;
    int       i, imax;
    TID_TYPE  tid ;

    /*  read remaining lines of file  */
    while (fgets (linebuf, MAXLINE, infile) != NULL)
        {
         imax = sscanf(linebuf, "%s %s %s %s %s %s %s %s %s %s %s %s %s %s %s",
                  ick[0],ick[1],ick[2],ick[3],ick[4],ick[5],ick[6],ick[7],
                  ick[8],ick[9],ick[10],ick[11],ick[12],ick[13],ick[14]);
         i=1;

/*  process PLACE line  */

if (! strncasecmp (ick[0], "PLACE", 5))
          {strcpy(sym_name,ick[i++]);

    if ((indx = find_place (sym_name)) >= 0)
```

```
        {
                place[indx].act_name = (char *)malloc(strlen(ick[i])+1);
                strcpy(place[indx].act_name,ick[i++]);
                strcpy(owner,ick[i++]);
tid = find_owner (owner) ;
place[indx].owner = tid ;
        }
    else
recInfo (globTid, "Could not find place '%s' by name\n",
                                        sym_name);
    }

/*  process TRANS line  */

else if (! strncasecmp (ick[0], "TRANS", 5))
            {strcpy(sym_name,ick[i++]);

             strcpy(buff,ick[i++]);

     if (! strncasecmp (buff, "SINGLE", 6))
        {if ((indx = find_trans (sym_name)) >= 0)
                    {trans[indx].act_name = (char *)malloc(strlen(ick[i])+1);
                     strcpy(trans[indx].act_name,ick[i++]);
                     strcpy(owner,ick[i++]);
     tid = find_owner (owner) ;
     trans[indx].owner = tid ;
                    }
         else
    recInfo (globTid, "Could not find trans '%s' by name\n",
                                             sym_name) ;
                }
     else if (! strncasecmp (buff, "DOUBLE", 6))
        {if ((indx = find_trans (sym_name)) >= 0)
                   {trans[indx].act_name = (char *)malloc(strlen(ick[i])+1);
                    strcpy(trans[indx].act_name,ick[i++]);
                    strcpy(owner,ick[i++]);
    tid = find_owner (owner) ;
    trans[indx].owner = tid ;

                    trans[indx].act_end_name = (char *)malloc(strlen(ick[i])+1);
                    strcpy(trans[indx].act_end_name,ick[i++]);
                    strcpy(owner,ick[i++]);
    tid = find_owner (owner) ;
    trans[indx].end_owner = tid ;
        recInfo (globTid, "Trans %s is split: %s , %s \n", trans[indx].name,
                          trans[indx].act_name, trans[indx].act_end_name);
                   }
         else
   recInfo (globTid, "Could not find trans '%s' by name\n",
                                             sym_name) ;
                };
    }

/*  process DISPLAY line  */

else if (! strncasecmp (ick[0], "DISPLAY", 7))
   {net_name = (char *) malloc (strlen(ick[i])+1) ;
    strcpy (net_name, ick[i++]) ;
            if (i < imax)
       {playTid = find_owner(ick[i]);
              }
              else
                 playTid = globTid; /* Default action */
   };

/*  ignore all other lines  */

}
    return (OK) ;
    }

/*****************************************************************************
  find_owner  - find tid of owner of place or transition
****************************************************************************/
TID_TYPE  find_owner (char *owner)
    {
/*
     want to add "smarter" function than remembers already found owners
*/
    return (msgTidQuery (globTid, owner)) ;
    }
```

C.8 *msgs.c* — Sends Various Messages

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>

#include <stdio.h>

#include <ctos.h>
#include <btsLib.h>

#include "Global_def.h"
#include "msg.h"
#include "Global_var.p"

extern TID_TYPE toTid;
extern char *myName ;

#define  FIRE_RATE  (1)

/************************************************************************
*   disp_set_tok()
*      This routine builds and sends a message to the display
*     indicating the new contents of a place.
*/

void  disp_set_tok (int p_num)
    {
    UPDATES  *pu ;
    MSG_TYPE  msg ;

    /*  for local place, send msgs to displays  */
    if (place[p_num].owner == globTid)
        {
/*  get start of updates list  */
if ((pu = place[p_num].update) == NULL)
    return ;
```

```
/*  build message to displays  */
msgBuild (&msg, 0, globTid, MSG_SET_TOK, NULL, MS_NONE, MF_STANDARD) ;

/*  loop through list  */
while (pu)
    {
    msg.dest = pu->tid ;
    msg.data = (void *) ((pu->num & 0x0000ffff)
 | (place[p_num].curr_token << 16)) ;
    msgSend (&msg) ;
/*
    recInfo (globTid, "Set tokens=%2i in place '%s.%s' on display %x\n",
     place[p_num].curr_token, myName, place[p_num].name,
     pu->tid) ;
*/
    pu = pu->next ;
    }

/*        recInfo (globTid, "Set tokens=%2i in place '%s.%s'\n",
 place[p_num].curr_token, myName, place[p_num].name) ; */
        }


    /*  for remote place, send msg to owning player  */
    else
        {
msgBuildSend (place[p_num].owner, globTid, MSG_ADD_TOKEN,
      (void *) ( ((FIRE_RATE << 16) & 0xffff0000)
                | INDEX_FROM_ID(place[p_num].id) ),
      MS_NONE, MF_STANDARD) ;

/*        recInfo (globTid, "%s: Add token to remote place '%s' in net %x\n",
 myName, place[p_num].name, place[p_num].owner) ; */
}
    }

/************************************************************************
*   disp_start_fire()
*      This routine builds and sends a message to the display
*    indicating that a transition has begun firing.
*/

void  disp_start_fire (int t_num)
    {
    UPDATES  *pu ;
    MSG_TYPE  msg ;

    /*  get start of updates list  */
    if ((pu = trans[t_num].update) == NULL)
        return ;

    /*  build message to displays  */
    msgBuild (&msg, 0, globTid, MSG_START_TRANS, NULL, MS_NONE, MF_STANDARD) ;

    /*  loop through list  */
    while (pu)
        {
        if ((pu->which)&(T_START))
            {
   msg.dest = pu->tid ;
   msg.data = (void *) pu->num ;
   msgSend (&msg) ;
/*
   recInfo (globTid, "Start firing trans '%s.%s' on display %x\n",
    myName, trans[t_num].name, pu->tid) ;
*/
            };
pu = pu->next ;
         }

    recInfo (globTid, "Start firing trans '%s.%s'\n", myName, trans[t_num].name);

    }

/***************************************************************************
*   disp_end_fire()
*      This routine builds and sends a message to the display
*    indicating that a transition has ended firing.
*/

void  disp_end_fire (int t_num)
    {
    UPDATES  *pu ;
    MSG_TYPE  msg ;

    /*  get start of updates list  */
    if ((pu = trans[t_num].update) == NULL)
        return ;

    /*  build message to displays  */
    msgBuild (&msg, 0, globTid, MSG_END_TRANS, NULL, MS_NONE, MF_STANDARD) ;

    /*  loop through list  */
    while (pu)
        {
        if ((pu->which)&(T_END))
           {
   msg.dest = pu->tid ;
   msg.data = (void *) pu->num ;
   msgSend (&msg) ;
/*
   recInfo (globTid, "End firing trans '%s.%s' on display %x\n",
    myName, trans[t_num].name, pu->tid) ;
*/
            };
pu = pu->next ;
         }

    recInfo (globTid, "End firing trans '%s.%s'\n", myName, trans[t_num].name);

    }
```

C.9 *var_man.c* — Manages the Variables for Local Functions

```
#include "Global_def.h"

struct var_entry var_table[MAX_VAR];

int find_variable(char *name)
```

```
{
 int i;

 i = 0;

 while ((i<MAX_VAR)&&(var_table[i].ident_str))
    {if (!strcmp(name,var_table[i].ident_str))
       return(i);
     i++;
    };

 if (i<MAX_VAR)
    {var_table[i].ident_str = (char *)malloc(strlen(name)+1);
     strcpy(var_table[i].ident_str,name);
     return(i);
    };

 return(-1);
}

int read_var(int index)
{
 return(var_table[index].value);
}

void save_var(int index, int value)
{
 var_table[index].value = value;
}

char *get_var_name(int index)
{
 return(var_table[index].ident_str);
}
```

APPENDIX D
Display Software

D.1 *Makefile* — The Display Makefile

```
# Makefile for test of UNIX version of CTOS

INC1 = /home/page/MCS/src/lib/btsLib
INC2 = /home/page/MCS/src/lib/msgLib
INC3 = /home/page/MCS/src/lib/ctosShell
INC4 = /usr2/testbed/CIRSSE/installed/UNIX/h
INC5 = /home/lefebvre/DEMOS/pnets/player
INC6 = /home/page/MCS/src/lib/recLib
INC7 = /home/lefebvre/unix/ctos/domLib

CFLAGS = -O -Wreturn-type -Wunused -Wswitch -Wcomment
         -Wshadow -Wpointer-arith -DMAKE_SUN4 -I$(INC1) -I$(INC2)
         -I$(INC3) -I$(INC4) -I$(INC5) -I$(INC6) -I$(INC7) -DOS_UNIX

BINDIR = /home/page/MCS/bin/sun4/
BTSLIB = $(BINDIR)btsErr.o $(BINDIR)btsSem.o $(BINDIR)btsGlobals.o
         $(BINDIR)msgLib.o $(BINDIR)msgBCMem.o
PLAYDIR = /home/lefebvre/DEMOS/pnets/player/
RECDIR = /home/page/MCS/bin/sun4/

# List of targets...
# ----------------

all : disp_eh


# E.H. Tasks...
# ------------

disp_eh:  disp_eh.o Xsetup.o Xdraw.o manip_net.o monitor.o Xinter.o \
  $(PLAYDIR)load_net.o  $(PLAYDIR)var_man.o
gcc $(CFLAGS) -o disp_eh disp_eh.o Xsetup.o Xdraw.o manip_net.o \
monitor.o Xinter.o $(PLAYDIR)tape.o $(PLAYDIR)load_net.o    \
$(RECDIR)recLib.o $(PLAYDIR)enable_func.o $(PLAYDIR)post_func.o \
$(PLAYDIR)var_man.o $(BTSLIB) $(BINDIR)ctosShell.o \
-L/home/lefebvre/DEMOS/demo1 -L/home/lefebvre/unix/ctos/domLib \
-lcoord -ldom -lX11 -lm

disp_eh.o: disp_eh.c  $(PLAYDIR)Global_def.h
gcc -c $(CFLAGS) disp_eh.c -o disp_eh.o

Xsetup.o: Xsetup.c  $(PLAYDIR)Global_def.h
gcc -c $(CFLAGS) Xsetup.c -o Xsetup.o

Xdraw.o: Xdraw.c  $(PLAYDIR)Global_def.h
gcc -c $(CFLAGS) Xdraw.c -o Xdraw.o

manip_net.o: manip_net.c  $(PLAYDIR)Global_def.h
gcc -c $(CFLAGS) manip_net.c -o manip_net.o

monitor.o: monitor.c
gcc -c $(CFLAGS) monitor.c -o monitor.o
```

```
Xinter.o: Xinter.c  $(PLAYDIR)Global_def.h
gcc -c $(CFLAGS) Xinter.c -o Xinter.o
```

D.2 *disp_eh.c* — The Display Event Handler

```
/*****************************************************************************
    File:     disp_eh.c

    Purpose:  "Petri Net Display" event handler displays portions of net

    Author:   Joe Peck
              revised:  Don Lefebvre

    Revised:  20 Aug'91
*****************************************************************************/
/*------------------------------------ include files -------------------*/
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <stdio.h>
#include <signal.h>

#include <ctos.h>
#include <btsLib.h>
#include <recLib.h>

#include "msg.h"
#include "Global_def.h"
#include "Global_var.h"

/*------------------------------------ global variables ----------------*/
char   root_name[200];
char   buff[200];
char  *myName ;
char   config_file [MAXLINE] ;

/*------------------------------------ function prototypes --------------*/
STATUS  load_net          (char *net_config_file, int type) ;
STATUS  register_updates () ;
void    printPTinfo       () ;
void    go_monitor        ();

MyTask(disp_eh)


/****************************************************************************
  disp_eh  - subnet display for Petri Net
-----------------------------------------------------------------------------
  SYMBOLIC NAME:  (used to configure net)

  RETURNS:        0 indicating processing complete

  PARAMETERS:     TID_TYPE  myTid - task id of event handler
                  MSG_TYPE *msg   - pointer to received message
****************************************************************************/
int disp_eh(TID_TYPE myTid, MSG_TYPE *msg)
    {
```

```
    static BOOL  ready = FALSE ;
    static int proc_id;
    int k, l;
    char databuff[10];

    switch(msg->command)
        {
        case MSG_TASK_ARGS:
             strcpy(buff, msg->data);
             break;


case MSG_PINIT:
             globTid = myTid;
     myName = msgMyTaskName (myTid) ;

             sscanf(buff,"%s %s", root_name, display_name);

     /*  read display config file  */
             if (load_net (root_name, CF_DISP) == ERROR)
         {
 recInfo (myTid, "Unable to load config file '%s'.disp",
  root_name) ;
 msgAckAINIT (myTid) ;
         msgApplicationExit (myTid) ;
         }
             recInfo (myTid  "DISP: Loaded net '%s.disp'\n", root_name);

     /*  create X window  */
             Xinit (myName, NULL, 0) ;
             recInfo (myTid, "%s Initialized window\n", myName) ;

             /*  initialize variables  */
             visible_layer = 0xffff;
             tags_on = 1;
             scale = 1.0;
             offset_x = 0.0;
             offset_y = 0.0;


     /*  scale the net  */
             rescale_net(scale, offset_x, offset_y);
        /*     recInfo (myTid, "%s Scaled net\n", myName) ; */
             sleep(2);

     /*  display the net  */
             draw_whole_net();
             recInfo (myTid, "%s Displayed net\n", myName) ;
     /* ready = TRUE ; */

             break ;


case MSG_AINIT:
     register_updates () ;

             /*  Fork off the process to monitor input  */
             proc_id = fork();
             if (proc_id == 0)    /* monitor process */
                go_monitor();
             break ;


case MSG_AEXEC:
             recInfo (myTid, "%s Received AEXEC\n", myName) ;
/*      printPTinfo () ;  */
             break ;


case MSG_DRAW_WHOLE_NET:
             recInfo (myTid, "%s Received DRAW_WHOLE_NET\n", myName) ;
             draw_whole_net();
             break;


        case MSG_SIT_ON_HANDS:
             recInfo (myTid, "%s Received SIT_ON_HANDS\n", myName) ;
             sleep(5);
             break;


        case MSG_SET_TOK:
/*             recInfo (myTid, "%s Received SET_TOK\n", myName) ; */
             set_token(msg->source, (int)msg->data);
             break;


        case MSG_START_TRANS:
/*             recInfo (myTid, "%s Received START_TRANS\n", myName) ; */
             mark_trans(msg->source, (int)msg->data);
             break;


        case MSG_END_TRANS:
/*             recInfo (myTid, "%s Received END_TRANS\n", myName) ; */
             unmark_trans(msg->source, (int)msg->data);
             break;


        case MSG_TOGGLE_TAGS:
/*             recInfo (myTid, "%s Received TOGGLE_TAGS\n", myName) ; */
             tags_on ^= 1;
             draw_whole_net();
             break;


        case MSG_VISIBLE:
/*             recInfo (myTid, "%s Received VISIBLE\n", myName) ; */
             visible_layer = (int)msg->data;
             draw_whole_net();
             break;


        case MSG_SHIFT_DISPLAY:
/*             recInfo (myTid, "%s Received SHIFT_DISPLAY\n", myName) ; */
             offset_x += (float)HORI_SHIFT((int)msg->data);
             offset_y += (float)VERT_SHIFT((int)msg->data);
             rescale_net(scale, offset_x, offset_y);
             draw_whole_net();
             break;
```

```
          case MSG_CHANGE_SCALE:
/*               recInfo (myTid, "%s Received CHANGE_SCALE\n", myName) ; */
               scale += (float)((int)(msg->data)*SCALE_FACTOR);
               if (scale <= 0.0)
                  scale = 0.1;
               rescale_net(scale, offset_x, offset_y);
               draw_whole_net();
               break;


          case MSG_SELECT_PLACE:
               recInfo (myTid, "%s Received SELECT_PLACE\n", myName) ;
               l = GET_INDEX((int)msg->data);
               switch(GET_BUTTON((int)msg->data))
                  {case 1:
                      recInfo(globTid,"Should add a token\n");
                        msgBuildSend(place[l].owner, myTid,
                           MSG_ADD_TOKEN,
                           (void *) ((INDEX_FROM_ID(place[l].id)
                              & 0x0000ffff) | (1 << 16)),
                           MS_NONE, MF_STANDARD);
                      break;
                   case 2:
                      recPrompt(myTid,databuff,"Place %s: Enter token quantity: ",
                                              place[l].name);
                      if (sscanf(databuff,"%d",&k))
                         if (k>=0)
                            msgBuildSend(place[l].owner, myTid,
                              MSG_SET_TOK,
                              (void *) ((INDEX_FROM_ID(place[l].id)
                                 & 0x0000ffff) | (k << 16)),
                              MS_NONE, MF_STANDARD);
                      break;
                   case 3:
                      recInfo(globTid,"Should subtract a token\n");
                        msgBuildSend(place[l].owner, myTid,
                           MSG_SUB_TOKEN,
                           (void *) ((INDEX_FROM_ID(place[l].id)
                              & 0x0000ffff) | (1 << 16)),
                           MS_NONE, MF_STANDARD);
                      break;
                  };
               break;

          case MSG_SELECT_TRANS:
               recInfo (myTid, "%s Received SELECT_TRANS\n", myName) ;
               l = GET_INDEX((int)msg->data);
               if (mode == SINGLE_STEP)
                  {msgBuildSend(trans[l].owner, myTid,
                        MSG_CHK_FIRE_TRANS,
                        (void *)INDEX_FROM_ID(trans[l].id),
                        MS_NONE, MF_STANDARD);
                  };
               break;

          case MSG_SET_MODE:
               recInfo (myTid, "%s Received SET_MODE\n", myName) ;
               mode = (int)msg->data;
               break;
```

```
        case MSG_PTERM:
/*              recInfo (myTid, "%s Received PTERM\n", myName) ; */
             kill(prcc_id,SIGKILL);
             break;
        }

    /*  until X window refresh works, redraw everything  */
    if (ready == TRUE)
       {
/*        draw_whole_net();   */
        usleep(500) ;
       };

    return (msgDefaultProc (myTid, msg)) ;
    }


/********************************************************************************
  printPTinfo  - print place & transition information
********************************************************************************/
void  printPTinfo ()
    {
    int  i ;

    if (!strncasecmp (myName, "DISP_A", 6))
        sleep (3) ;
    else if (!strncasecmp (myName, "DISP_B", 6))
        sleep (4) ;
    else if (!strncasecmp (myName, "DISP_C", 6))
        sleep (5) ;

    recInfo (globTid, "\nNet Name: %s\n", myName) ;

    recInfo (globTid, "PLACES [%i]\n", num_pl) ;
    for (i=0; i<num_pl; i++)
        {
recInfo (globTid, "\t%6s  %8x  %4x \n", place[i].name,
 place[i].id, place[i].owner) ;
}

    recInfo (globTid, "TRANSITIONS [%i]\n", num_tr) ;
    for (i=0; i<num_tr; i++)
        {
recInfo (globTid, "\t%6s  %8x  %4x  \n", trans[i].name,
 trans[i].id, trans[i].owner) ;
}

    }


/********************************************************************************
  register_updates  - sends request msgs to register display updates
********************************************************************************/
STATUS  register_updates ()
    {
    int          i ;
    REG_REMOTE  *ps ;

    /*  send msg for each place  */
    for (i=0; i<num_pl; i++)
        {
```

```
ps = (REG_REMOTE *) malloc (sizeof (REG_REMOTE)) ;
strcpy (ps->sym_name, place[i].name) ;
strcpy (ps->act_name, place[i].act_name) ;
ps->type = PLACE_NODE ;
ps->id   = i ;
       place[i].id = msgBuildSend (place[i].owner, globTid, MSG_REGISTER_UPDATE,
    (void *) ps, sizeof (REG_REMOTE),
    MF_REPLYWAIT) ;
        }

    /*  send msg for each trans  */
    for (i=0; i<num_tr; i++)
        {
ps = (REG_REMOTE *) malloc (sizeof (REG_REMOTE)) ;
strcpy (ps->sym_name, trans[i].name) ;
strcpy (ps->act_name, trans[i].act_name) ;
ps->type = TRANS_NODE ;
ps->id   = i ;
        if (trans[i].end_owner == NULL)
           {ps->which = T_BOTH;
            trans[i].id = msgBuildSend (trans[i].owner, globTid,
                                   MSG_REGISTER_UPDATE,
    (void *) ps, sizeof (REG_REMOTE)
    MF_REPLYWAIT) ;
           }
        else
           {ps->which = T_START;
            trans[i].id = msgBuildSend (trans[i].owner, globTid,
                                   MSG_REGISTER_UPDATE,
    (void *) ps, sizeof (REG_REMOTE),
    MF_REPLYWAIT) ;

    ps = (REG_REMOTE *) malloc (sizeof (REG_REMOTE)) ;
            strcpy (ps->sym_name, trans[i].name) ;
            strcpy (ps->act_name, trans[i].act_end_name) ;
    ps->type = TRANS_NODE ;
    ps->id   = i ;
            ps->which = T_END;
            trans[i].end_id = msgBuildSend (trans[i].end_owner, globTid,
                                   MSG_REGISTER_UPDATE,
    (void *) ps, sizeof (REG_REMOTE),
    MF_REPLYWAIT) ;
           };
        }

    return (OK) ;
    }
```

D.3 *Xdraw.c* — Performs All X-Drawing Actions

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xcs.h>
#include <X11/Xatom.h>

#include <stdio.h>

#include <ctos.h>
#include <btsLib.h>
```

```
#include "recLib.h"
#include "Global_def.h"
#include "msg.h"

#include "Xdraw.p"

extern Display *display;
extern Window window;
extern GC gc;
extern XFontStruct *act_font;

extern struct Place *place;
extern struct Trans *trans;
extern int num_pl, num_tr;
extern int radius, radius2, diameter;
extern int tags_on, rates_on, visible_layer;
/*
extern struct tape_node *head;

extern char *find_tape_name();

struct tape_entry
   {int id;
    char *ident_str;
   };

struct tape_node
   {int id;
    struct tape_node *next;
   };
*/


/**********************************************************************
*   draw_whole_net()
*      This routine clears the display window and redraws the
*    Petri net.  Layers are dealt with by the called routines.
*/

void draw_whole_net(void)
{
 int i;

 XClearWindow(display, window);

/*
 draw_tape();
*/

/* printf("Total places: %d\n",num_pl); */
/* printf("Total transitions: %d\n",num_tr); */

 for (i=0;i<num_tr;i++)
    {draw_trans(i);
     if (tags_on)
        draw_trans_tag(i);
     if (rates_on)
        draw_trans_rate(i);
     draw_arcs(i);
     if (trans[i].firing == TRUE)
        highlight_trans(i);
    };

 for (i=0;i<num_pl;i++)
    {draw_place(i); /* this is reduntant: draw_token redraws the place */
     draw_token(i);
     if (tags_on)
        draw_place_tag(i);
    };
 XFlush(display);
}

/************************************************************************
*   set_token()
*      This routine sets the number of tokens in a place and
*    redisplays the place.
*/

void set_token(TID_TYPE from, int data)
{
 int p_num, num_tok;

 num_tok = (data&0xffff0000)>>16;
 p_num = data&0xffff;

 place[p_num].curr_token = num_tok;
 draw_token(p_num);
 XFlush(display);
}

/************************************************************************
*   mark_trans()
*      This routine highlights a transition to indicate that
*    it has started to fire.
*/

void mark_trans(TID_TYPE from, int data)
{
 int t_num;

 t_num = data&0xffff;
 if (++trans[t_num].firing == 1)
    {highlight_trans(t_num);
     XFlush(display);
    };
}

/************************************************************************
*   unmark_trans()
*      This routine unhighlights a transition to indicate that
*    a transition has completed firing.
*/

void unmark_trans(TID_TYPE from, int data)
{
 int t_num;

 t_num = data&0xffff;
 if (trans[t_num].firing != 0)
    if (--trans[t_num].firing == 0)  /* just reached zero */
       {highlight_trans(t_num);
        XFlush(display);
       };
}

/************************************************************************
*   valid_layer()
*      This routine checks if any of the indicated layers are
*    currently enabled for display.
*/

int valid_layer(int layer)
{
 return (visible_layer & layer);
}

/************************************************************************
*   draw_place()
*      This routine draws the indicated place.  As a side effect,
*    it erases any tokens that were already drawn.
*/

void draw_place(int i)
{
 if (valid_layer(place[i].layer))
    {draw_type(ERASE);
     XFillArc(display, window, gc, place[i].x_loc - radius,
                                   place[i].y_loc - radius,
                                   diameter, diameter, 0, 360*64);

     draw_type(DRAW);
     XDrawArc(display, window, gc, place[i].x_loc - radius,
                                   place[i].y_loc - radius,
                                   diameter, diameter, 0, 360*64);
/* printf("Drawing place %d at %d,%d\n",i,place[i].x_loc,place[i].y_loc); */
   };
}

/************************************************************************
*   draw_token()
*      This routine redraws the indicated place to erase the old
*    tokens, then redraws all of the tokens in the place.  Up to
*    nine tokens may be drawn, thereafter digits are used.  The
*    token size is changed based on the number of tokens in the place.
*/

void draw_token(int i)
{
 char buff[5];

 draw_place(i);

 if (valid_layer(place[i].layer))
    {switch(place[i].curr_token)
        {case 0: break;
         case 1:
            draw_1_tok(place[i].x_loc, place[i].y_loc,L);
            break;

         case 2:
            draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc,L);
            draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc,L);
            break;
```

```
case 3:
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc + radius/2,L);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc + radius/2,L);
    draw_1_tok(place[i].x_loc, place[i].y_loc - radius/2,L);
    break;

case 4:
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc + radius/2,M);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc + radius/2,M);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc - radius/2,M);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc - radius/2,M);
    break;

case 5:
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc + radius/2,M);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc + radius/2,M);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc - radius/2,M);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc - radius/2,M);
    draw_1_tok(place[i].x_loc, place[i].y_loc,M);
    break;

case 6:
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc - radius/2,S);
    break;

case 7:
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc,S);
    break;

case 8:
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc,S);
    break;

case 9:
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc + radius/2,S);
    draw_1_tok(place[i].x_loc, place[i].y_loc - radius/2,S);
    draw_1_tok(place[i].x_loc - radius/2, place[i].y_loc,S);
    draw_1_tok(place[i].x_loc + radius/2, place[i].y_loc,S);
```

```
                draw_1_tok(place[i].x_loc, place[i].y_loc,S);
                break;

            default: sprintf(buff, "%d", place[i].curr_token);
                     XDrawString(display, window, gc,
                      place[i].x_loc - XTextWidth(act_font, buff, strlen(buff))/2,
                      place[i].y_loc + (act_font->ascent)/2,
                                              buff, strlen(buff));
                   break;
          };
      };
}

/*************************************************************************
*   draw_1_token()
*      This routine draws a single token at the given location and
*     size.
*/

void draw_1_tok(int x,int y, float size)
{
   XFillArc(display, window, gc, x - max((int)(radius*size),2)/2,
                             y - max((int)(radius*size),2)/2,
                             max((int)(radius*size),2), max((int)(radius*size),2),
                             0, 360*64);
}

/*************************************************************************
*   draw_place_tag()
*      This routine draws the tag associated with the given place.
*/

void draw_place_tag(int i)
{
 if (valid_layer(place[i].layer))
    XDrawString(display, window, gc, place[i].tag_x, place[i].tag_y,
                     place[i].name, strlen(place[i].name));
}

/*************************************************************************
*   draw_trans()
*      This routine draws the given transition.  The line type used
*     indicates the type of transition, and the correct orientation
*     is used as well.
*/

void draw_trans(int i)
{
 if (valid_layer(trans[i].layer))
    {set_line(trans[i].kind);

     switch(trans[i].orient)
        {case 0:  XDrawLine(display, window, gc, trans[i].x_loc - radius,
                      trans[i].y_loc, trans[i].x_loc + radius, trans[i].y_loc);
                  break;
         case 1:  XDrawLine(display, window, gc, trans[i].x_loc,
                    trans[i].y_loc - radius, trans[i].x_loc,
                    trans[i].y_loc + radius);
                  break;
         case 2:  XDrawLine(display, window, gc,
                    trans[i].x_loc - radius2,trans[i].y_loc - radius2,
                    trans[i].x_loc + radius2,trans[i].y_loc + radius2);
                  break;
          default:
                  break;
         };
      set_line(DEFAULT);
     };
}

/****************************************************************************
*   draw_trans_tag()
*      This routine draws the tag associated with the given transition.
*/

void draw_trans_tag(int i)
{
 if (valid_layer(trans[i].layer))
    XDrawString(display, window, gc, trans[i].tag_x, trans[i].tag_y,
                     trans[i].name, strlen(trans[i].name));
}

/****************************************************************************
*   draw_trans_rate()
*      This routine draws the rate associated with the given transition.
*/

void draw_trans_rate(int i)
{
 char buff[MAXLINE];

 if (valid_layer(trans[i].layer))
    {sprintf(buff, "%f", trans[i].fire_rate);
     XDrawString(display, window, gc, trans[i].rate_x, trans[i].rate_y,
                     buff, strlen(buff));
    };
}

/****************************************************************************
*   highlight_trans()
*      This routine draws an inverted circle over the given transition,
*    changing it's current display state.
*/

void highlight_trans(int i)
{
 if (valid_layer(trans[i].layer))
    {draw_type(INVERT);
     XFillArc(display, window, gc, trans[i].x_loc - radius,
            trans[i].y_loc - radius,
            diameter, diameter, 0, 360*64);
     draw_type(DRAW);
    };
}

/****************************************************************************
*   draw_arcs()
*      This routine draws the input, output and inhibitor arcs for
*    the given transition.
*/

void draw_arcs(int i)
```

```
{
 struct Arc *arc_a;
 struct Arc_point *arc_point_a;
 int x, y;

 arc_a = trans[i].inp_arc;
 while (arc_a)
    {if (valid_layer(arc_a->layer))
        {x = arc_a->x_start;
         y = arc_a->y_start;
         arc_point_a = arc_a->arc_points;
         while(arc_point_a)
            {XDrawLine(display, window, gc, x, y,
                 arc_point_a->x, arc_point_a->y);
             x = arc_point_a->x;
             y = arc_point_a->y;
             arc_point_a = arc_point_a->next;
            };
         XDrawLine(display, window, gc, x, y, arc_a->x_end, arc_a->y_end);
         XDrawLine(display, window, gc, arc_a->arrow->x0, arc_a->arrow->y0,
                                        arc_a->arrow->x1, arc_a->arrow->y1);
         XDrawLine(display, window, gc, arc_a->arrow->x0, arc_a->arrow->y0,
                                        arc_a->arrow->x2, arc_a->arrow->y2);
         XDrawLine(display, window, gc, arc_a->arrow->x1, arc_a->arrow->y1,
                                        arc_a->arrow->x2, arc_a->arrow->y2);
        };
     arc_a = arc_a->next;
    };

 arc_a = trans[i].out_arc;
 while (arc_a)
    {if (valid_layer(arc_a->layer))
        {x = arc_a->x_start;
         y = arc_a->y_start;
         arc_point_a = arc_a->arc_points;
         while(arc_point_a)
            {XDrawLine(display, window, gc, x, y,
                    arc_point_a->x, arc_point_a->y);
             x = arc_point_a->x;
             y = arc_point_a->y;
             arc_point_a = arc_point_a->next;
            };
         XDrawLine(display, window, gc, x, y, arc_a->x_end, arc_a->y_end);
         XDrawLine(display, window, gc, arc_a->arrow->x0, arc_a->arrow->y0,
                                        arc_a->arrow->x1, arc_a->arrow->y1);
         XDrawLine(display, window, gc, arc_a->arrow->x0, arc_a->arrow->y0,
                                        arc_a->arrow->x2, arc_a->arrow->y2);
         XDrawLine(display, window, gc, arc_a->arrow->x1, arc_a->arrow->y1,
                                        arc_a->arrow->x2, arc_a->arrow->y2);
        };
     arc_a = arc_a->next;
    };

 arc_a = trans[i].inh_arc;
 while (arc_a)
    {if (valid_layer(arc_a->layer))
        {x = arc_a->x_start;
         y = arc_a->y_start;
         arc_point_a = arc_a->arc_points;
         while(arc_point_a)
            {XDrawLine(display, window, gc, x, y,
                  arc_point_a->x, arc_point_a->y);
            x = arc_point_a->x;
            y = arc_point_a->y;
            arc_point_a = arc_point_a->next;
           };
        XDrawLine(display, window, gc, x, y, arc_a->x_end, arc_a->y_end);
        XFillArc(display, window, gc, arc_a->arrow->x0 - radius/3,
                                      arc_a->arrow->y0 - radius/3,
                     diameter/3, diameter/3, 0, 360*64);
       };
    arc_a = arc_a->next;
   };
}

#define LEFT_EDGE 10
#define TOP_EDGE  10
#define WIDTH     1024
#define HEIGHT    30
#define MAX_CHAR  10

/*
draw_tape()
{
 struct tape_node *tmp;
 char *name;
 int offset, height, shift, i=1;

 draw_type(ERASE);
 XFillRectangle(display, window, gc, LEFT_EDGE, TOP_EDGE-1,
                                        WIDTH, HEIGHT);
 draw_type(DRAW);

 XDrawString(display, window, gc, LEFT_EDGE, TOP_EDGE+act_font->ascent,
                    "TAPE:", 5);

 tmp = head;
 shift = act_font->max_bounds.width;
 offset = MAX_CHAR*act_font->max_bounds.width;
 height = act_font->max_bounds.ascent+act_font->max_bounds.descent;

 while(tmp)
    {name = find_tape_name(tmp->id);
     XDrawRectangle(display, window, gc, LEFT_EDGE+(i)*offset-shift,
                         TOP_EDGE-1, offset, height+2);

     XDrawString(display, window, gc, LEFT_EDGE+(i++)*offset,
                                 TOP_EDGE+act_font->ascent,
                                 name, strlen(name));
     tmp = tmp->next;
    };
}
*/
```

D.4 *Xsetup.c* — Initializes the Windowing System

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
```

```
#include <stdio.h>

#include "recLib.h"
#include "Global_def.h"
#include "Global_var.p"

#define BORDER 3

/*---------------------------- function prototypes --------------------*/
void  getGC(Window window, GC *gc);
void  load_font(void);
void  set_font(int i);


/************************************************************************
*   Xinit()
*      This routine initializes the X-variables and opens
*    a window.
*/

void  Xinit(char *nets_name, char *argv[], int argc)
{
 int x, y;
 unsigned int width, height, display_width, display_height;
 char *window_name = nets_name;
 char *icon_name = nets_name;
 XSizeHints size_hints;
 XWMHints wm_hints;
 XClassHint class_hints;
 XTextProperty windowname,iconname;
 char *program_name = "Display";


/* connect to X server */
 if ((display = XOpenDisplay(display_name)) == NULL)
    {recInfo(globTid,"Net %s cannot open display %s\n", nets_name,
                                       XDisplayName(display_name));
     exit(-1);
    };

/* get screen information */
 screen_num = DefaultScreen(display);
 display_width = DisplayWidth(display, screen_num);
 display_height = DisplayHeight(display, screen_num);

/* set desired window settings */
 x = 0;
 y = 0;
 width = display_width/2;
 height = display_height/2;

/* create the window */
 window = (Window *)XCreateSimpleWindow(display,
                  RootWindow(display,screen_num), x, y,
                  width, height, BORDER, BlackPixel(display, screen_num),
                  WhitePixel(display, screen_num));

/* provide hints for the window manager */
 size_hints.flags = PPosition | PSize;

/* setup the window and icon name */
```

```
 if (XStringListToTextProperty(&window_name, 1, &windowname) == 0)
    {fprintf(stderr,"Net %s failed structure allocation for windowname.\n",
                              nets_name);
     exit(-1);
    };

 if (XStringListToTextProperty(&icon_name, 1, &iconname) == 0)
    {fprintf(stderr,"Net %s failed structure allocation for iconname.\n",
                              nets_name);
     exit(-1);
    };

 wm_hints.initial_state = NormalState;
 wm_hints.input = True;
 wm_hints.flags = StateHint | InputHint;

 class_hints.res_name = program_name;
 class_hints.res_class = program_name;

 XSetWMProperties(display, (int)window, &windowname, &iconname,argv,argc,
                     &size_hints, &wm_hints, &class_hints);

/* set desired event types */
 XSelectInput(display, (int) window, ExposureMask | KeyPressMask |
                        ButtonPressMask | StructureNotifyMask);

/* load fonts */
 load_fonts();

/* create GC */
 getGC(window, &gc);

/* display the window */
 XMapWindow(display, (int) window);

 XFlush(display);
}

/****************************************************************************
*   getGC()
*      This routine creates the graphics context (GC) and
*    sets up the default parameters.
*/

void  getGC(Window window, GC *gc)
{
 unsigned long valuemask = 0;
 XGCValues values;
 int line_style = LineSolid;
 int cap_style = CapRound;
 int join_style = JoinRound;
 int dash_offset = 0;

 *gc = XCreateGC(display, window, valuemask, &values);

 set_font(1);

 XSetFont(display, *gc, act_font->fid);

 XSetForeground(display, *gc, BlackPixel(display,screen_num));
 XSetLineAttributes(display, *gc, line_width, line_style,
                              cap_style, join_style);

}

/*****************************************************************************
*   set_line()
*      This routine changes the line type depending upon
*    the type of transition to be drawn.
*/

void  set_line(int i)
{
 switch(i)
    {case IMM:  XSetLineAttributes(display, gc, line_width*3/2, LineSolid,
                         CapButt, JoinRound);
               break;
     case DET:  XSetLineAttributes(display, gc, radius/2, LineSolid,
                         CapButt, JoinRound);
               break;
     case EXP:  XSetLineAttributes(display, gc, radius/2, LineSolid,
                         CapButt, JoinRound);
               break;
     default:  XSetLineAttributes(display, gc, line_width, LineSolid,
                         CapButt, JoinRound);
    };
}

/*****************************************************************************
*   load_fonts()
*      This routine loads all of the fonts that the display
*    needs to show tokens, tags, etc.
*/

void  load_fonts()
{
 if (!(font[0] = XLoadQueryFont(display,"5x8")))
    {fprintf(stderr,"Cannot load 5x8 font\n");
     exit(-1);
    };

 if (!(font[1] = XLoadQueryFont(display,"6x10")))
    {fprintf(stderr,"Cannot load 6x10 font\n");
     exit(-1);
    };

 if (!(font[2] = XLoadQueryFont(display,"8x13")))
    {fprintf(stderr,"Cannot load 8x13 font\n");
     exit(-1);
    };

 if (!(font[3] = XLoadQueryFont(display,"9x15")))
    {fprintf(stderr,"Cannot load 9x15 font\n");
     exit(-1);
    };
}

/*****************************************************************************
*   set_font()
*      This routine changes the current font in use.
*/
```

```
void  set_font(int i)
{
 act_font = font[i];
 XSetFont(display, gc, act_font->fid);
}

/****************************************************************************
*   draw_type()
*      This routine changes the drawing mode.
*/

void  draw_type(int mode)
{
 switch(mode)
    {case ERASE:  XSetForeground(display, gc, WhitePixel(display,screen_num));
                  XSetFunction(display, gc, GXcopy);
                  break;
     case DRAW:   XSetForeground(display, gc, BlackPixel(display,screen_num));
                  XSetFunction(display, gc, GXcopy);
                  break;
     case INVERT: /* XSetForeground(display,gc, WhitePixel(display,screen_num));
                  XSetFunction(display, gc, GXxor); */
                  XSetFunction(display, gc, GXinvert);
                  break;
     default:     break;
    };
}
```

D.5 *manip_net.c* — Performs Scaling and Shifting of Display

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <stdio.h>
#include <math.h>

#include "Global_def.h"
#include "Global_var.p"


modify_end_point(arc,x,y)
   struct Arc *arc;
   int x,y;
{
 int dx, dy;
 double angle;

 dx = x - arc->x_end;
 dy = y - arc->y_end;

 if (dx == 0)
    {if (dy >0)
        arc->y_end += radius;
      else
        arc->y_end -= radius;
     return;
    };
```

```
  if (dy == 0)
     {if (dx >0)
         arc->x_end += radius;
       else
         arc->x_end -= radius;
      return;
     };

  angle = atan((double)dy/dx);
  if (dx > 0)
     {arc->x_end += radius*cos(angle);
      arc->y_end += radius*sin(angle);
     }
   else
     {arc->x_end -= radius*cos(angle);
      arc->y_end -= radius*sin(angle);
     };

}

modify_arrow(arc,x,y)
   struct Arc *arc;
   int x,y;
{
 int dx, dy, x0, y0;
 double angle;
 struct Arrowhead *arr;

 if (arc->arrow)
    arr = arc->arrow;
  else
    {arr = (struct Arrowhead *)malloc(sizeof(struct Arrowhead));
     arc->arrow = arr;
    };

 if (arc->type == OUTARC)
    {x0 = arc->x_end;
     y0 = arc->y_end;
    }
  else
    {x0 = arc->x_start;
     y0 = arc->y_start;
    };

arr->x0 = x0;
arr->y0 = y0;

dx = x - x0;
dy = y - y0;

if (dx == 0)
   {if (dy >0)
       angle = -M_PI/2;
     else
       angle = M_PI/2;
   }
else
   {if (dy == 0)
      angle = 0.0;
    else
      angle = atan((double)dy/dx);
     };

  if (dx > 0)
     {arr->x1 = x0 + radius*cos(angle+ANG);
      arr->y1 = y0 + radius*sin(angle+ANG);
      arr->x2 = x0 + radius*cos(angle-ANG);
      arr->y2 = y0 + radius*sin(angle-ANG);
     }
   else
     {arr->x1 = x0 - radius*cos(angle+ANG);
      arr->y1 = y0 - radius*sin(angle+ANG);
      arr->x2 = x0 - radius*cos(angle-ANG);
      arr->y2 = y0 - radius*sin(angle-ANG);
     };

}

modify_end_circle(arc)
   struct Arc *arc;
{
 int x, y;
 struct Arrowhead *arr;

 arr = arc->arrow;
 arr->x0 = (2*arr->x0 + arr->x1 + arr->x2)/4;
 arr->y0 = (2*arr->y0 + arr->y1 + arr->y2)/4;
}

rescale_net(scale,offset_x,offset_y)
   float scale, offset_x, offset_y;
{
 int i, x, y, x1, y1;
 struct Arc *arc_a, *arc_b;
 struct Arc_point *arc_point_a, *arc_point_b;

 diameter = (int)DIAMETER*scale;
 radius = diameter/2;
 radius2 = (int)(radius/1.414);

 for (i=0;i<num_mp;i++)
    {mpar[i].tag_x = X_IN_TO_PIX(mpar[i].tag_x_in);
     mpar[i].tag_y = Y_IN_TO_PIX(mpar[i].tag_y_in);
    };

 for (i=0;i<num_rp;i++)
    {rpar[i].tag_x = X_IN_TO_PIX(rpar[i].tag_x_in);
     rpar[i].tag_y = Y_IN_TO_PIX(rpar[i].tag_y_in);
    };

 for (i=0;i<num_pl;i++)
    {place[i].x_loc = X_IN_TO_PIX(place[i].x_loc_in);
     place[i].y_loc = Y_IN_TO_PIX(place[i].y_loc_in);
     place[i].tag_x = X_IN_TO_PIX(place[i].tag_x_in);
     place[i].tag_y = Y_IN_TO_PIX(place[i].tag_y_in);
    };

 for (i=0;i<num_tr;i++)
    {trans[i].x_loc = X_IN_TO_PIX(trans[i].x_loc_in);
     trans[i].y_loc = Y_IN_TO_PIX(trans[i].y_loc_in);
     trans[i].tag_x = X_IN_TO_PIX(trans[i].tag_x_in);
     trans[i].tag_y = Y_IN_TO_PIX(trans[i].tag_y_in);
```

```
trans[i].rate_x = X_IN_TO_PIX(trans[i].rate_x_in);
trans[i].rate_y = Y_IN_TO_PIX(trans[i].rate_y_in);

arc_a = trans[i].inp_arc;
while(arc_a)
   {x = arc_a->x_start = trans[i].x_loc;
    y = arc_a->y_start = trans[i].y_loc;
    arc_a->x_end = place[arc_a->place].x_loc;
    arc_a->y_end = place[arc_a->place].y_loc;
    if (arc_point_a = arc_a->arc_points)
       {x1 = X_IN_TO_PIX(arc_point_a->x_in);
        y1 = Y_IN_TO_PIX(arc_point_a->y_in);
       }
     else
       {x1 = arc_a->x_end;
        y1 = arc_a->y_end;
       };
    while(arc_point_a)
       {x = arc_point_a->x = X_IN_TO_PIX(arc_point_a->x_in);
        y = arc_point_a->y = Y_IN_TO_PIX(arc_point_a->y_in);
        arc_point_a = arc_point_a->next;
       };
    modify_end_point(arc_a,x,y);
    modify_arrow(arc_a,x1,y1);
    arc_a = arc_a->next;
   };

arc_a = trans[i].out_arc;
while(arc_a)
   {x = arc_a->x_start = trans[i].x_loc;
    y = arc_a->y_start = trans[i].y_loc;
    arc_a->x_end = place[arc_a->place].x_loc;
    arc_a->y_end = place[arc_a->place].y_loc;
    arc_point_a = arc_a->arc_points;
    while(arc_point_a)
       {x = arc_point_a->x = X_IN_TO_PIX(arc_point_a->x_in);
        y = arc_point_a->y = Y_IN_TO_PIX(arc_point_a->y_in);
        arc_point_a = arc_point_a->next;
       };
    modify_end_point(arc_a,x,y);
    modify_arrow(arc_a,x,y);
    arc_a = arc_a->next;
   };

arc_a = trans[i].inh_arc;
while(arc_a)
   {x = arc_a->x_start = trans[i].x_loc;
    y = arc_a->y_start = trans[i].y_loc;
    arc_a->x_end = place[arc_a->place].x_loc;
    arc_a->y_end = place[arc_a->place].y_loc;
    if (arc_point_a = arc_a->arc_points)
       {x1 = X_IN_TO_PIX(arc_point_a->x_in);
        y1 = Y_IN_TO_PIX(arc_point_a->y_in);
       }
     else
       {x1 = arc_a->x_end;
        y1 = arc_a->y_end;
       };
    while(arc_point_a)
       {x = arc_point_a->x = X_IN_TO_PIX(arc_point_a->x_in);
        y = arc_point_a->y = Y_IN_TO_PIX(arc_point_a->y_in);
```

```
                  arc_point_a = arc_point_a->next;
                 };
              modify_end_point(arc_a,x,y);
              modify_arrow(arc_a,x1,y1);
              modify_end_circle(arc_a);
              arc_a = arc_a->next;
             };
    };

 line_width = 0;
 if (diameter < 20)
    set_font(0);
  else
     if (diameter < 26)
        set_font(1);
      else
         if (diameter < 35)
            {set_font(2);
             line_width = 2;
            }
          else
             {set_font(3);
              line_width = 3;
             }
}
```

D.6 *monitor.c* — Processes X-Window Events

```
#include <X11/Xlib.h>
#include <X11/Xutil.h>
#include <X11/Xos.h>
#include <X11/Xatom.h>
#include <X11/keysym.h>
#include <stdio.h>

#include <ctos.h>

#include "Global_def.h"
#include "Global_var.p"
#include "msg.h"

extern Display *display;
extern radius;

void go_monitor()
{
 int id, dist;
 XEvent report;
 XKeyEvent *rep_ptr;
 char buff[5];
 char buf[10];
 KeySym key;
 int f;
 MSG_TYPE msg;
 int shift_hor=0, shift_ver=0, size_chng=0, rescale = FALSE;

      while (1)  {
                XNextEvent(display, &report);
                switch  (report.type) {
                case Expose:
```

```
                /* unless this is the last contiguous expose,
                 * don't draw the window */
                if (report.xexpose.count != 0)
                        break;

                msgBuildSend (globTid, globTid, MSG_DRAW_WHOLE_NET,
                        NULL, MS_NONE, MF_STANDARD);
                break;

        case ButtonPress:
                rep_ptr = (struct XKeyEvent *)&report;
                near_trans(rep_ptr->x,rep_ptr->y,&id,&dist);
                if (dist <= radius)
                   msgBuildSend(globTid, globTid, MSG_SELECT_TRANS,
                    ENCODE_BUTTON(report.xbutton.button,id),
                    MS_NONE, MF_STANDARD);
                near_place(rep_ptr->x,rep_ptr->y,&id,&dist);
                if (dist <= radius)
                   msgBuildSend(globTid, globTid, MSG_SELECT_PLACE,
                    ENCODE_BUTTON(report.xbutton.button,id),
                    MS_NONE, MF_STANDARD);
                break;

        case KeyPress:
                f = XLookupString(&report, buff, 5, &key, 0);
                switch(key)
                   {case XK_t:
                    case XK_T:
                           msgBuildSend(globTid, globTid,
                                   MSG_TOGGLE_TAGS,
                                   NULL, MS_NONE, MF_STANDARD);
                           break;

                    case XK_s:
                          if (globTid != playTid)
                      {recInfo(globTid,"Sending local mode: %d\n",
                                                      SINGLE_STEP);
                          msgBuild(&msg, globTid, globTid, MSG_SET_MODE,
                            (void *)SINGLE_STEP, MS_NONE, MF_STANDARD);
                          msgSend(&msg);
                          msgBuild(&msg, playTid,globTid, MSG_SET_MODE,
                            (void *)SINGLE_STEP, MS_NONE, MF_STANDARD);
                          msgSend(&msg);
                         };
                          break;
                    case XK_S:
                      recInfo(globTid,"Sending global mode: %d\n",
                                                      SINGLE_STEP);
                          msgBuild(&msg, globTid, globTid, MSG_SET_MODE,
                            (void *)SINGLE_STEP, MS_NONE, MF_STANDARD);
                          msgSend(&msg);
                          msgBroadcast(&msg, MB_APPLICATION);
                          break;
                    case XK_c:
                          if (globTid != playTid)
                       {recInfo(globTid,"Sending local mode: %d\n",
                                                        CONTINUOUS);
                          msgBuild(&msg, globTid,globTid, MSG_SET_MODE,
                            (void *)CONTINUOUS, MS_NONE, MF_STANDARD);
                          msgSend(&msg);
                          msgBuild(&msg, playTid,globTid, MSG_SET_MODE,
            (void *)CONTINUOUS, MS_NONE, MF_STANDARD);
          msgSend(&msg);
         };
          break;
    case XK_C:
       recInfo(globTid,"Sending Global mode: %d\n",
                                               CONTINUOUS);
          msgBuild(&msg, globTid, globTid, MSG_SET_MODE,
            (void *)CONTINUOUS, MS_NONE, MF_STANDARD);
          msgSend(&msg);
          msgBroadcast(&msg, MB_APPLICATION);
          break;
    case XK_r:
    case XK_R:
           msgBuildSend (globTid, globTid,
              MSG_DRAW_WHOLE_NET, NULL,
              MS_NONE, MF_STANDARD);
           break;
    case XK_Left:
           msgBuildSend(globTid, globTid,
             MSG_SHIFT_DISPLAY,
             (void *)LEFT, MS_NONE, MF_STANDARD);
           shift_hor = LEFT;
           rescale = TRUE;
           break;

    case XK_Right:
           msgBuildSend(globTid, globTid,
             MSG_SHIFT_DISPLAY,
             (void *)RIGHT, MS_NONE, MF_STANDARD);
           shift_hor = RIGHT;
           rescale = TRUE;
           break;

    case XK_Up:
           msgBuildSend(globTid, globTid,
             MSG_SHIFT_DISPLAY,
             (void *)UP, MS_NONE, MF_STANDARD);
           shift_ver = UP;
           rescale = TRUE;
           break;

    case XK_Down:
           msgBuildSend(globTid, globTid,
             MSG_SHIFT_DISPLAY,
             (void *)DOWN, MS_NONE, MF_STANDARD);
           shift_ver = DOWN;
           rescale = TRUE;
           break;

    case XK_comma:
    case XK_less:
           msgBuildSend(globTid, globTid,
             MSG_CHANGE_SCALE,
             (void *)DEC, MS_NONE, MF_STANDARD);
           size_chng = DEC;
           rescale = TRUE;
           break;

    case XK_period:
    case XK_greater:
```

```
                                        msgBuildSend(globTid, globTid,
                                          MSG_CHANGE_SCALE,
                                          (void *)INC, MS_NONE, MF_STANDARD);
                                        size_chng = INC;
                                        rescale = TRUE;
                                        break;

                                 case XK_q:
                                 case XK_Q:
                                        recPrompt(globTid, buf,
                                              "Confirm Application Exit (y/n): ");
                                        if ((buf[0] == 'y')||(buf[0] == 'Y'))
                                           msgApplicationExit (globTid) ;
                                        break;

                                 case XK_0:
                                 case XK_1:
                                 case XK_2:
                                 case XK_3:
                                 case XK_4:
                                 case XK_5:
                                 case XK_6:
                                 case XK_7:
                                 case XK_8:
                                 case XK_9:
                                      visible_layer ^= 1<<(key-XK_0);
                                      msgBuildSend(globTid, globTid, MSG_VISIBLE,
                                        (void *)visible_layer, MS_NONE, MF_STANDARD);
                                      break;
                                };

                      default:
                              break;
                      } /* end switch */

                if (rescale)
                   {offset_x += (float)HORI_SHIFT(shift_hor);
                    offset_y += (float)VERT_SHIFT(shift_ver);
                    scale += (float)(size_chng*SCALE_FACTOR);
                    if (scale <= 0.0)
                       scale = 0.1;
                    rescale_net(scale, offset_x, offset_y);

                    shift_hor = shift_ver = size_chng = 0;
                   };

        } /* end while */
}
```

D.7 *Xinter.c* — User Selection of Places and Transitions

```
#include <stdio.h>
#include <math.h>

#include "Global_def.h"

extern struct Trans *trans;
extern struct Place *place;
extern int num_tr;
extern int num_pl;

/*************************************************************************
*   near_trans()
*      This routine locates the transition nearest the point x,y.  The
*    variables id and dist are modified to reflect the id number
*    of the transition and the distance to it in pixels.
*/

near_trans(x,y,id,dist)
   int x,y,*id,*dist;
{
 int min, i;

 min = (trans[0].x_loc-x)*(trans[0].x_loc-x) +
                      (trans[0].y_loc-y)*(trans[0].y_loc-y);
 *id = 0;

 for (i=1;i<num_tr;i++)
    {*dist = (trans[i].x_loc-x)*(trans[i].x_loc-x) +
                      (trans[i].y_loc-y)*(trans[i].y_loc-y);
     if (*dist < min)
        {min = *dist;
         *id = i;
        };
    };
 *dist = (int)sqrt((double)min);
}

/*************************************************************************
*   near_place()
*      This routine locates the place nearest the point x,y.  The
*    variables id and dist are modified to reflect the id number
*    of the place and the distance to it in pixels.
*/

near_place(x,y,id,dist)
   int x,y,*id,*dist;
{
 int min, i;

 min = (place[0].x_loc-x)*(place[0].x_loc-x) +
                      (place[0].y_loc-y)*(place[0].y_loc-y);
 *id = 0;

 for (i=1;i<num_pl;i++)
    {*dist = (place[i].x_loc-x)*(place[i].x_loc-x) +
                      (place[i].y_loc-y)*(place[i].y_loc-y);
     if (*dist < min)
        {min = *dist;
         *id = i;
        };
    };
 *dist = (int)sqrt((double)min);
}
```