A PETRI NET ON-LINE CONTROLLER FOR THE COORDINATION OF MULTIPLE MOBILE ROBOTS

FAUSTINA HWANG

A PETRI NET ON-LINE CONTROLLER FOR THE COORDINATION OF MULTIPLE MOBILE ROBOTS

by

©Faustina Hwang, B.Eng.

A thesis submitted to the

School of Graduate Studies

in partial fulfillment of the

requirements for the degree of

Master of Engineering

Faculty of Engineering and Applied Science
Memorial University of Newfoundland

August 2000

St. John's, Newfoundland, Canada

Abstract

In applications such as mining, space exploration, and toxic waste cleanup, mobile robots are often required to move within a common environment and to share resources. This introduces the need for a means of coordinating their behaviours. Also, due to the unpredictable nature of the worksite, there is a need to accommodate changes in a dynamic environment.

A general framework for group robotics was developed in response to this need. The framework includes a discrete event controller for on-line control and runtime monitoring, the focus of the current research.

A Petri net based discrete event formalism has been investigated as a basis for the development of an on-line controller. From a high-level task description, a set of rules have been used to automatically generate a Petri net structure that provides coordinated behaviour. The Petri net can then be executed to send instructions to robots and to incorporate feedback from the robots at runtime. This on-line controller has been used to control mobile robots in a proof-of-concept demonstration. In a laboratory setting, the Petri net controller was able to coordinate the behaviour of two robots in marker-based navigation tasks.

Although the work completed to date has provided promising results, many research challenges remain. Some suggestions for future work are presented.

Acknowledgements

Many thanks to Prof. Andy Fisher, Dr. Ray Gosine, Dr. Theo Norvell, and Dr. Siu O'Young for their guidance and encouragement. A thank you as well to Dr. Rokonuzzaman, Jamie King, and Rod Hale for countless hours of discussion and the sharing of ideas. A number of engineering co-op students have worked on parts of this project, and I thank each of them for their contributions. I would also like to thank the Faculty of Engineering and C-CORE, Memorial University of Newfoundland. Financial support from NSERC and the VP Special Initiatives Fund of Memorial University was greatly appreciated.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Why use autonomous mobile robots?

Mining, space exploration, forestry, underwater exploration, and toxic waste cleanup are but a few examples of areas in which the use of mobile robots can be of tremendous benefit. Autonomous and semi-autonomous robots can operate in dangerous environments and perform operations that are hazardous to humans. In this way, robotic systems can reduce the risk to human life. Many industrial tasks performed by humans can be slow and highly-repetitive, and because they require constant attention, can be very fatiguing. Automation of these tasks using robotic technology can reduce the cognitive load on workers.

1.2 Why use groups of robots?

There are a number of potential advantages to using groups of mobile robots rather than single robots. Certain tasks may be too complex or even impossible to be completed by a single robot. Constructing and using a number of simpler robots can be easier, cheaper, more flexible, and more fault-tolerant than using a single robot to

complete a task. Many robots can be in many places at the same time, and many robots can do many things at the same time. Multi-robot teams can take advantage of parallelism and redundancy to increase system robustness and achieve performance gains.

Thus, in many applications, semi-automated systems involving groups of mobile robots are a logical choice. In mining, multiple automated load-haul-dump vehicles can simultaneously travel between ore piles and crushers, resulting in high productivity and a reduced need for humans to perform time-consuming tasks[36]. In toxic waste cleanup, teams of mobile robots can be sent to waste sites to map the location of buried waste and to retrieve, sort, treat, and package the waste. In underwater applications, multiple machines can be deployed to inspect the hundreds of structural nodes of an offshore oil and gas platform, or to cooperate in the construction of a deepwater oil and gas facility[20].

1.3 Why is coordination necessary?

In many real-world applications, multiple robots are required to move within a common environment and to share resources (e.g. roads and intersections) without causing collision or deadlock. This is particularly important in certain industrial applications where vehicle collisions are not only dangerous, but may result in significant costs in terms of vehicle repair and production downtime. It has been recognized that the task of deploying mining trucks from a central garage requires a significant level of coordination among vehicles. In forestry applications where many vehicles are manually driven throughout a common worksite, collisions are not uncommon, but could be avoided with a formal method of coordinating their movements. This introduces the need for a means of detecting the possibility of collision and deadlock during the task planning and resource allocation phase (i.e. in simulation).

1.4 Why is simulation alone insufficient?

Many industrial applications would require mobile robots to operate in a semi-structured, dynamic environment. Due to the unpredictable nature of the worksite, simulation alone is insufficient to guarantee collision-free and deadlock-free operation. For example, in a mining application, it is difficult and sometimes impossible to know in advance the precise length of time required by a vehicle to traverse a tunnel. There is a possibility that the vehicle may encounter an obstacle, run out of fuel, or be required to drive over difficult terrain.

Due to these uncertainties, the state of the system at any given time is non-deterministic. It is therefore necessary to monitor the robots during operation and to send appropriate control signals at runtime. On-line system state monitoring can also be used to dynamically reschedule the robots to deal with changes in the operating conditions. In this way, it may be possible to optimize the operation of the system.

1.5 A Petri net approach

A Petri net based discrete event formalism has been investigated as a basis for the development of an on-line controller for multiple mobile robot systems. A discrete event system is a dynamic system that changes state in accordance with the abrupt occurrence of a physical event[35].

1.5.1 Why Petri net theory?

Petri net theory is well-suited to describing and studying systems characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic[30].. Systems of mobile robots can often exhibit a number of these characteristics. The theory also offers many formal analysis techniques. Petri nets can be formally verified

against the occurrence of potentially undesirable states (e.g. collision and/or deadlock). Rules exist to describe the dynamic behaviour of Petri nets, and thus, Petri nets can be executed at runtime to exhibit a specified behaviour. The potential then exists for a Petri net controller to generate control signals and to incorporate feedback from robots at runtime. In this way, it may be possible to achieve on-line monitoring of system states.

1.5.2 Why automatic Petri net generation?

Developing a Petri net model of a system and designing a Petri net controller requires an intricate knowledge of Petri net theory and its properties. Many of the tasks that are executed by robots in industrial applications, however, have to be specified by researchers or operators who may not have robot-specific knowledge, let alone a familiarity with Petri net modeling. The need for effective human-machine interfaces in robotic applications has been well-recognized [4, 17, 33].

In recognition of this need, a method of automatically generating a Petri net controller from a high-level task description has been investigated. It is envisioned that tasks requiring coordination of multiple robots will be specified at a high-level using a user-friendly, graphical interface. From this description, the Petri net controller is automatically generated according to constraints imposed by the working environment and by the rules of operation. In this way, it is possible to hide the details of Petri net theory from the operator, while still providing the ability to describe a robotic task formally and to analyze it.

Automatic Petri net generation also has the potential to accommodate changing operating conditions. It is envisioned that dynamic scheduling will be used to optimize the system during operation. A dynamic scheduler will consider changes in the composition of the robot team as well as changes in the operating environment

and allocate or re-allocate tasks to robots as necessary. The output of the dynamic scheduler would be a task description which, if translated automatically into a Petri net structure, has great potential for developing systems well-equipped to deal with runtime changes in a dynamic environment.

1.6 Contributions to research

Throughout the course of this research, the following contributions were made by the author:

- Development of a method for modeling road networks using a logical representation
- Development of a simple language for high-level specification of marker-based navigation tasks for robots
- Development of rules which can be used in the automatic generation of a Petri net controller
- Contributions to a Petri net software application initially developed at C-CORE:
    1. Incorporation of inhibitor arcs
    2. Capability to deal with conflict
    3. Capability to perform on-line control
    4. Capability to incorporate runtime feedback
    5. Capability to perform reachability analysis
    6. Capability to receive a task description from a separate software application and automatically generate a Petri net according to the rules mentioned previously

In addition, as a member of a small team of researchers (less than 6), the author made contributions to the development of a general framework for group robotics and to demonstrations illustrating the concept of Petri net on-line control.

1.7 Thesis Summary

Chapter 2 presents a summary of the literature in several related areas. In Chapter 3, a brief introduction to Petri net theory is given. Further elements of Petri net theory are introduced as they become relevant. In Chapter 4, a general framework for group robotics, conceived by members of the Faculty of Engineering at Memorial University of Newfoundland and the Intelligent Systems group at C-CORE, is presented to explain the context in which the Petri net controller was developed. Chapter 5 details the on-line Petri net controller and the methods used for automatic Petri net generation. In Chapter 6, some additional modules which were developed for testing purposes are described, as well as the results of experiments conducted using the Petri net controller. Future work and conclusions of this research are presented in Chapter 7.

Chapter 2

Related Work

A review of the literature is presented in the following related fields: cooperative mobile robots operating in dynamic environments, coordination of multiple robots to achieve collision- and deadlock-free behaviour, discrete event control of robotic tasks using Petri net theory, the application of Petri net theory to the field of mobile robotics, and Petri net-based controllers used in runtime execution and monitoring.

There has been some research into the development of control frameworks for cooperative mobile robots operating in dynamic environments. Parker's ALLIANCE architecture[34] addresses the issues of fault-tolerance, reliability, and adaptability for teams of mobile robots. The fault-tolerant response considered in the work is the dynamic re-selection (or re-allocation) of tasks due to robot failures or a dynamically changing environment. Adaptivity refers to the ability of the robot team to change its behaviour over time in response to changes to either improve performance or to prevent unnecessary degradation in performance. The outcome of this work is a mission planner that is dynamic, but the objective is not to deal with collision-avoidance. The experiments described deliberately avoid situations where the probability of collision is high.

Brumitt and Stentz[8] propose GRAMMPS, a Generalized Robotic Autonomous Mobile Mission Planning System for multiple mobile robots operating in unstructured environments. Again, however, they do not address inter-robot collision avoidance. Furthermore, this work considers applications where robots and goals are interchangeable (e.g. applications of an exploratory nature), and the concepts are not generally applicable.

Other areas of research address the coordination of multiple robots to achieve collision-free and deadlock-free behaviour. Alami et. al.[1] describe the MARTHA project. High-level missions are produced by a Central Station and sent to robots, which then use a Plan Merging Paradigm to communicate with all the other robots in the system to achieve coordinated behaviour. Noreils[32] developed a language used to describe Predicate/Transition nets which are executed to control coordinated protocols. One limitation of this approach is that coordinated protocols must be programmed and downloaded prior to execution, limiting the possibility of changing control strategies at run-time. Singh and Fujimura[37] suggest a navigation strategy that can be used to achieve cooperative behaviour among a set of mobile robots in tasks such as mapping of an unknown bounded region. Collision detection and avoidance is solved with a method of arbitration in individual cases of impending collision between robots.

Azarm and Schmidt[5] introduce a decentralized approach to achieving conflict-free motion, an approach involving a dynamic prioritization of the robots. In cases where the robot priority scheme fails, inter-robot communication and a method of negotiation are used to resolve the conflict. In this work, conflicts are seen as events, and are resolved as they occur, sometimes requiring some vehicles to backtrack in their routes.

Bourbakis[7] discusses the difficulty of achieving efficient synchronization of robots moving in a dynamic environment while avoiding collisions. He presents a generic

traffic priority language, called KYKLOFORIA, which is used by each robot to make decisions during navigation and avoid possible collisions with other moving objects.

Most of these works address collision avoidance at a local level, describing methods to be used by individual robots to resolve conflicts, often requiring substantial sensory information as well as communication with other robots in the environment. There has been little emphasis on coordinating the vehicles at a higher level, with the aim of preventing situations where impending collision becomes an issue. Some exceptions, however, are described in [14] and [10] which recognize the need to resolve potential conflicts before they can occur.

The work of Causse and Pampagnin[14] was carried out to develop a prototype transport system dealing with heavy loads in hospitals. The intended application implies a number of functional requirements, including the sharing of common resources such as elevators, corridors, and parking areas between robots. Also, the robots are required to navigate indoors along a known network of paths. Their approach recognizes the need for conflicts to be resolved before two robots can block each other, and performs traffic control by the booking of nodes in a topological graph that represents the current environment. Caloud et. al.[10] establish a set of behaviour rules which implement space allocation policies. All robots are required to communicate with each other as necessary to abide by the rules, thereby achieving coordination of the motions of multiple robots.

A detailed review of much of the existing work in cooperative mobile robotics can be found in [11].

There has been a significant amount of research into the discrete event control of robotic tasks using Petri net theory[16, 27, 31, 41, 12, 22, 26, 9, 24]. A large portion of the work is carried out in the context of Flexible Manufacturing Systems and robotic assembly tasks. Petri net models are frequently used in off-line simulation and analysis, and subsequently used to programme robots to perform tasks. In an

area of work not specific to robotic applications, there have been developments in translating Petri nets into control languages such as ladder logic[3, 39, 40]. Although Petri net theory appears to be valuable in off-line simulation, the need to programme robots to perform tasks remains a limitation in terms of developing systems capable of accommodating changing requirements during operation.

There have been a few reports on the application of Petri net theory to the field of mobile robotics. Causse and Christensen [13] present issues in control architectures for autonomous mobile robots, and express the view that any control architecture must mix several kinds of hierarchies. They then explain how hierarchical principles may be formulated in a single framework using Coloured Petri Net models. Montano et. al.[28] view control systems for mobile robots as a collection of concurrent processes: robot control, image processing, data from rangefinder processing, decision making, planning, etc. They use a time Petri net formalism to allow verification of functional and temporal system requirements, and also to allow automatic code generation, thereby avoiding coding mistakes. Petri nets were also used by Oliveira et. al.[33] as a formal language to describe the structure of the mission-control software of an autonomous underwater vehicle. The Petri net description was used in automatic code generation, an aspect which is discussed further below. Caloud et. al.[10] use hierarchical Petri nets to interpret plan decompositions and to monitor execution of tasks being carried out by mobile robots. The Petri net allows the robots to react to unexpected events.

Petri nets in mobile robotics has been largely applied to the control systems of single robots. Our approach differs in that we investigate the utility of the Petri net formalism in the higher-level task of coordinating the actions of multiple robots, rather than at the vehicle-level.

Petri net theory has, for a large part, been applied in off-line simulation. However, in many applications, the uncertain and dynamic nature of the working environment

makes it difficult to guarantee collision-free and deadlock-free operation through simulation alone. Rather, a method of runtime execution and monitoring is required. The use of Petri net theory for this type of on-line control has been mentioned in several works.

Caloud et. al.[10] present the GOFER project whose goal is to control the operations of many mobile robots in an indoor environment in order to automate a variety of tasks. Their system for planning and execution integrates task planning, task allocation, motion planning, and execution monitoring. The execution system uses a hierarchical Petri net formalism to monitor execution and react to unexpected events. Given an instance of a plan, a robot generates a net composed of states, action transitions, and hierarchy transitions. Action transitions correspond to the performance of an action, while hierarchy transitions have only a logical meaning in the process of plan interpretation. The planning and execution system is written in COMMON-LISP, and at the time of publication, experiments had only been performed with the help a simulator designed to simulate actions of autonomous agents.

Mascaro and Asada[26] present an approach to interactive control of human-robot systems using dual Petri nets. One Petri net represents the human side task process, while the other represents the robot side. They describe a proof-of-concept experiment involving a cable connection task which requires human-robot cooperation. The Petri net model of the task is translated into computer programs, using one computer to perform all actions pertaining to robot monitoring and control and a separate computer for monitoring the human. The control programs are written using an object oriented programming method where places, transitions, and tokens are represented by classes of objects which contain pointers connecting them to each other. Member functions are used to collect data, check conditions, and fire transitions.

Crockett et. al.[15] describe work in which a Petri net is used to describe the sequencing information for a manufacturing workstation. An application is composed

of a description of the Petri net model of the system using a declarative language, and action-causing procedures written in the "C" programming language. Their Petri Net-based Controller (PNC) runs on a general purpose computer, sends and receives ASCII messages, and works mainly with software interfaces. The PNC associates a procedure with each place and then executes that procedure when a token arrives at that place during Petri net execution.

The work of Freund and Rossman[17] uses an on-line Petri net monitor in the context of developing a virtual reality (VR) interface for robot control. The work was intended to make use of the capabilities of an already existing intelligent robot control system (IRCS), and to enhance the system with a VR interface. The IRCS was already capable of executing high level task descriptions. The challenge in the interface, then, was to translate the motions of a user into a series of tasks for the IRCS. A special class of Petri nets, "state/transition-nets with named marks", was used to monitor the events related to user actions in the virtual environment at run-time. The occurrence of an event would cause a state-change in the Petri net, which would result in an action being sent to the robot control system to be carried out at a particular time.

Lima et. al.[24] have developed a Petri-net-based application to coordinate the execution of robotic tasks and provide a human-machine interface. A robotic task can be designed through a graphical user interface, by drawing a Petri net and associating tasks to places and events to transitions. Task execution can be followed in real-time by watching the flow of tokens through the net. In their implementation, prior to execution, a designer is required to define in a file the location of the tasks which are used by the net. The software then takes care of directing the request to the appropriate location. Examples of applications to visual servoing and catching of moving objects by a robotic arm, and to mobile robot tasks are presented.

Oliveira et. al.[33], in their design of a mission control system for the MARIUS

autonomous underwater vehicle, have developed two specially designed software programming environments CORAL and ATOL. CORAL is a set of software tools that allows an operator to graphically build a library of elementary vehicle operations (vehicle primitives) embodied in Petri nets, and to run them in real time. ATOL provides similar tools for mission procedure programming. CORAL consists of two fundamental modules: the vehicle primitives library editor and generator, and the CORAL engine. The main goal of the first is to embody each vehicle primitive into a Petri net description. At runtime, the CORAL engine executes the Petri net and transition firings start the execution of tasks. The engine sends commands to and receives responses from the vehicle system tasks. This set-up allows for easy programming of missions, and also provides the system developer with a graphical user interface to monitor the state of progress of the mission based on the evolution of tokens in a Petri net.

The development of these Petri net on-line controllers has been, for the most part, dedicated to the control of single robots. In [10], although the intended application is a multi-robot system, the utility of the Petri net controller is not in the coordination of multiple robots. Rather, a set of behaviour rules is used along with inter-robot communication to deal with collision and deadlock. The present work explores the utility of a Petri net-based on-line controller in the coordination of multiple robots.

The present work also investigates an additional layer of automation which has the potential to facilitate the development of Petri nets used for task execution and monitoring. A method of automatically generating Petri nets to control tasks requiring cooperation of multiple robots is described. The tasks are described at a high-level, and the construction of the Petri net model is accomplished automatically. To the author's best knowledge, there has been no published work in this area. In [10], it is reported that "a robot generates a net", but details of how the net is generated are not given.

Chapter 3

Introduction to Petri Nets

3.1 Petri net theory

Petri nets, originating in the 1962 dissertation of Carl Adam Petri, are a graphical and mathematical modeling tool which can be applied to many systems. An introduction to Petri net theory is presented here. A complete tutorial-review on Petri nets can be found in [30].

A Petri net is a directed, weighted bipartite graph whose nodes are either *places* or *transitions*. Graphically, places are drawn as circles, and transitions are drawn as bars. Directed *arcs* are drawn both from places to transitions and from transitions to places. Arcs are labeled with weights (positive integers), and a k–weighted arc may be interpreted as the equivalent of k parallel arcs with unity weight. Each place may contain zero or more *tokens*, and each token is drawn as a black dot within the place. The *marking* of a Petri net indicates the number of tokens contained within each place, and is represented as an m–vector where m is the total number of places in the net. Formally, a Petri net is defined as a 5-tuple $PN = (P, T, A, W, M_o)$ where:

$P = \{p_1, p_2, \ldots, p_m\}$ is a finite set of places,

$T = \{t_1, t_2, \ldots, t_n\}$ is a finite set of transitions,

$A \subseteq (P \times T) \cup (T \times P)$ is a set of arcs,

$W : A \rightarrow \{1, 2, 3, \ldots\}$ is a weight function, and

$M_o : P \rightarrow \{0, 1, 2, 3, \ldots\}$ is the initial marking.

$P \cap T = \emptyset$ and $P \cup T \neq \emptyset$

Each transition has *input places* and *output places*. Formally, the set of input places of a transition t is given by $I(t) = \{p|(p,t)\epsilon A\}$. The set of output places of a transition t is given by $O(t) = \{p|(t,p)\epsilon A\}$.

In using Petri nets to model a task, one interpretation of the net components uses *conditions* and *events*. Places represent conditions and transitions represent events. Input places represent conditions which must be true before the event may occur. Output places represent conditions which are true after the event has occurred. When a condition is true, a token appears in the appropriate place. Other interpretations of Petri net components are also used, and can be found throughout the literature. For example, in some interpretations, the presence of tokens in a place does not represent the truth of a condition, but rather k tokens can represent that k items or resources are available Thus, a Petri net may be used to model resource allocation.

Using the Petri net components as presented above, system *behaviours* are modeled by applying the Petri net transition (firing) rule. A transition t is said to be *enabled* if each input place contains at least $w(p,t)$ tokens, where $w(p,t)$ is the weight of the arc connecting the input place p to transition t. Formally, t is enabled if $M(p) > w(p,t)$ for all $p\epsilon I(t)$. Once the transition is enabled, it may or may not fire, depending on whether or not the event actually takes place. The firing of a transition causes a change in marking by removing $w(p,t)$ tokens from each input place and adding $w(t,p)$ tokens to each output place, where $w(t,p)$ is the

weight of the arc connecting t to output place p. The new marking is M' where $M'(p) = M(p) - w(p,t) + w(t,p)$.

A transition without input places does not consume tokens, but acts as a source of tokens for its output places. Thus, it is called a *source transition*. Source transitions are always enabled. A transition without output places is called a *sink transition*; it consumes tokens from its input places but does not produce any. A single token can be removed from a place by only one transition; they are indivisible.

Example: Figure 3.1(a) shows a simple Petri net with six places and five transitions. All arcs are of unity weight. Initially, place p_1 is marked with a single token and t_1 is the only enabled transition. When it fires, the token is removed from p_1 and a token is placed in each of p_2 and p_3. At this point, both t_2 and t_3 are enabled and can fire *concurrently*. After t_2 and t_3 complete their firing, p_4 and p_5 each contain one token(Figure 3.1(b)). Transitions t_4 and t_5 are then in *conflict*—both transitions are enabled, but the firing of either disables the other.

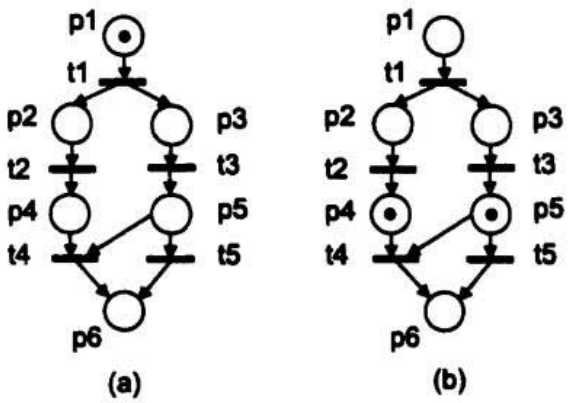


Figure 3.1: A Petri net example illustrating *concurrency* and *conflict*.

An *inhibitor arc* is a special type of arc, represented graphically as an arc whose arrow head has been replaced with a circle. If a place p is connected to a transition t by an inhibitor arc, the firing of t is inhibited by the presence of one or more tokens in p. When p is unmarked, it has no effect on the enabling and firing of t.

Example: Figure 3.2 illustrates the use of an inhibitor arc. In (a), transition $t1$ is enabled and fires. Following the firing (Figure 3.2(b)), a token is placed in p2 which inhibits any further firings of $t1$.

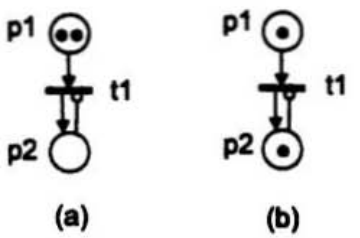


Figure 3.2: A Petri net example illustrating the use of inhibitor arcs.

Petri nets may be divided into two categories according to the firing characteristics of the transitions[42]. In *untimed* Petri nets, the transitions are considered to fire instantaneously. *Timed* Petri nets, on the other hand, contain transitions which require a certain amount of time to fire. During this time, tokens are neither in the input places nor output places, but rather are considered to be contained *within* the transition. Tokens emerge from the transition and are placed in the output places after the time of the transition has passed. How long this will be depends on the type of transition, whether it is *deterministic* or *stochastic*.

Deterministic transitions have a fixed firing time t. Each time the transition is

enabled, it requires t time units to fire. This type of transition is useful in representing a fixed task which requires a known length of time to be completed. *Stochastic transitions* have firing times which are selected each time they are enabled based on a probability distribution. Thus, the firing time may vary each time the transition is enabled. Stochastic transitions are useful for modeling unpredictable events.

The basic concepts of Petri net theory have been presented to explain the fundamental rules for Petri net execution. One of the major strengths of the Petri net formalism, however, is the support available for the analysis of many properties and problems associated with concurrent systems. A discussion of some behavioural properties (properties which depend on the initial marking of the net) is presented later (see Section 5.5), along with a discussion of analysis methods.

3.2 Petri nets and group robotics

Petri nets exhibit a number of properties which make them an attractive alternative for modeling systems of mobile robots. In particular, Petri net theory is able to accommodate some of the challenges presented by systems which require coordinated behaviour among robots and which operate in unstructured or semi-structured environments. The following advantages are noted:

1. Petri nets are naturally oriented towards the modeling and analysis of discrete event systems that are characterized as being concurrent, asynchronous, distributed, parallel, nondeterministic, and/or stochastic.

2. Petri nets can be used to model systems where the transitions between events are enabled according to arbitrarily complex rules.

3. Petri net based formalisms provide a systematic method for decomposing high-level behaviours (e.g. a complete task) into lower-level behaviours (e.g. a simple

autonomous task).

4. Petri net based formalisms provide a means for decomposing or modularizing potentially complex systems. Combining multiple systems can often be reduced to keeping several original nets unaltered, and adding a few places and/or transitions to achieve proper coupling.

5. Petri net theory provides well-developed, formal analysis methods which can be valuable in detecting potentially undesirable system behaviours (e.g. collision and/or deadlock).

6. Petri nets can be executed, thereby providing the potential for the development of on-line controllers capable of runtime execution and monitoring. Furthermore, their clear, graphical representation provides a means for developers to easily track system states by following the movement of tokens through the net.

Petri net controllers have been proposed as one of the components of a general framework for group robotics. This framework is presented in the following chapter.

Chapter 4

A General Framework for Group Robotics

The development of a Petri net-based on-line controller began in the context of on-going work toward a general framework for group robotics. This framework has been designed for the control of multiple mobile vehicles in an unstructured or semi-structured environment. In order to be truly useful in real-world applications, control architectures for multiple robot systems must explicitly address the dynamic nature of the robot team and its environment. A general framework was developed in response to this need.

The general framework for group robotics is shown in Figure 4.1. The components of interest at this time are the task description, the resource description, the dynamic scheduler, the discrete event controller, and the mobile robots. Details of the other components can be found in [19].

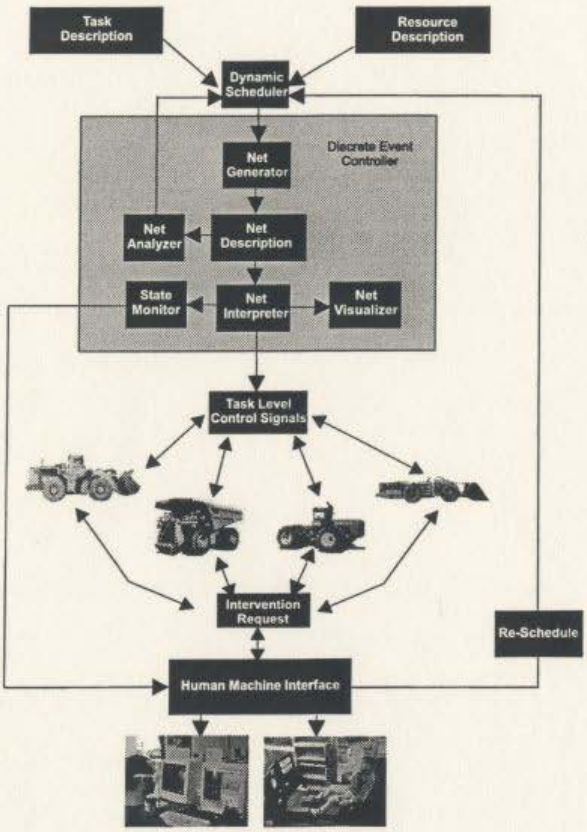


Figure 4.1: A general framework for group robotics.

4.1 Task Description

An operations planner describes robotic tasks as a combination of high-level subtasks. For example, the task for a load-haul-dump (LHD) vehicle may be to load at ore pile A, haul to a crusher at B, dump at the crusher, and to repeat the sequence while there is still ore at A. These high-level subtasks are expected to be completed semi-autonomously by the vehicles' onboard computing systems. It is anticipated that tasks will be described using an intuitive interface (e.g. graphical user interface), then input to a dynamic scheduler.

4.2 Resource Description

A description of system resources is provided to the dynamic scheduler. Resources (e.g. mining vehicles, roads, intersections, ore piles and crushers) are characterized by certain parameters. For example, links of roads would have associated costs that may vary for different types of vehicles, depending on factors such as fuel cost and fuel efficiency for the vehicle. The parameters are updated as the resources and the elements of the operating environment change.

4.3 Dynamic Scheduler

The role of the dynamic scheduler is to optimize the execution of the system (e.g. by maximizing efficiency). Taking into account that certain robots may only perform certain tasks, and also that robots capable of performing the same tasks may complete them differently, the dynamic scheduler must assign tasks to robots appropriate to the capabilities and performance of each robot. For example, the dynamic scheduler might have to select which load-haul-dump (LHD) vehicles to use for a task, given that some LHD's have greater capacity than others but also higher fuel costs. Furthermore, given that the mine environment is such that there are multiple paths which could be taken from the ore pile to the crusher, the dynamic scheduler would designate a route for each LHD, bearing in mind that optimal operation may require a minimization of total distance traveled.

4.4 Discrete Event Controller

The completion of a subtask by a robot can be considered a discrete event. A Petri net based discrete event formalism has been proposed as a basis for the development

of an on-line controller capable of coordinating multiple mobile robots.

4.4.1 Petri Net Generator

The output of the dynamic scheduler (and the input to the Petri net generator) is a description of the subtasks to be performed by each robot. For example, the dynamic scheduler may provide the generator with a list of robots to be used and also the road segments to be followed by each robot. The generator would then merge the individual robotic task descriptions into a centralized control scheme which can be formally verified to ensure coordinated behaviour among the multiple robots. For example, given that multiple LHD's are required to share roads and intersections in the mine, the role of the generator is to create a control scheme that is guaranteed not to result in collision or deadlock. A method of centralized control is proposed to achieve this coordination with minimal, if any, inter-robot communication.

4.4.2 Petri Net Interpreter

The output of the Petri net generator is a description of a Petri net controller (PNC). The Petri net can be executed to send task-level control signals to robots. From the perspective of the PNC, it is assumed that a single subtask (one command) can be successfully executed by the robot, either autonomously or through some method transparent to the PNC. The PNC tells each robot what to do, not how. For example, upon execution, the PNC could command a particular robot to "Move to point A". It assumes that the robot is capable of carrying out this fundamental task, and whether the robots moves to point A by following a light-line or through remote teleoperation is irrelevant. In this way, the PNC remains independent of robot architecture; the framework allows for the system to incorporate and take advantage of the latest advances in robot development.

Given that multi-robot teams are often required to operate in semi-structured environments, the state of the system at any point is non-deterministic. There is too much uncertainty in the system (e.g. determining where two robots will meet and when) in order to solve every possible robot-interaction in advance. In order to accommodate this, it is proposed that interactions be managed at execution time. The PNC is not only responsible for system execution, but also for runtime monitoring. Upon completion of each task, a robot notifies the PNC which is then able to update the state of the system. The PNC is then able to send a new command to the robot appropriate to the new system state.

4.5 Dynamic Re-scheduling

In the event that the operating conditions of the system change in such a way that the behaviour of the group of robots is affected (e.g. a robot breaks down, a particular road in a network must be closed), the dynamic scheduler reassigns the tasks to the robots (e.g. gives the tasks of the broken robot to another functioning robot, re-route the robots to avoid the newly closed road), and a new PNC is created.

The proposed framework is intended to provide the basis for the development of a system that is responsive to changes in individual robot skills and performance, to dynamic changes in the environment, and to changes in robot team composition while providing coordinated behaviour among multiple mobile vehicles. The framework minimizes inter-robot communication (a feature which is expected to enhance system scalability), emphasizes formal analysis methods for task validation, and has the potential to optimize system operation. Furthermore, this framework recognizes that dynamic operating conditions and other limitations prevent complete automation of all robotic tasks. Thus, it accommodates on-demand human-machine cooperation

and dynamic reconfigurability.

This thesis describes initial developments in one component of the framework, the discrete event controller. Work completed toward the development of a Petri net generator and a Petri net interpreter is presented.

Chapter 5

A Petri Net On-Line Controller

A software application has been developed at C-CORE and the Faculty of Engineering and Applied Science, Memorial University of Newfoundland which allows Petri net models to be created and executed. With this software, a model may be created in one of two ways:

1. The places, transitions (deterministic and stochastic), and arcs (normal and inhibitor) can be drawn by a user through the Graphical User Interface.

2. Provided with a task description and a resource description, the software application can automatically generate a Petri net without user input.

In the proposed framework for group robotics, a dynamic scheduler provides a Petri net generator with a description of high-level subtasks to be performed by each robot. The generator then merges the individual task descriptions into a centralized control scheme. Currently, *marker-based navigation tasks within a network of roads* are supported, and a Petri net controller can be automatically generated to ensure proper sharing of roads and intersections.

In order to accomplish this, the Petri net generator requires (a) information about the nature of the operating environment (e.g. which roads are connected), and (b) a

set of rules which govern how resources are shared (e.g. only one robot is permitted in an intersection at any time). The environment representation and the principles of resource sharing are presented next.

5.1 Environment Modeling

The generation of a control scheme that ensures proper sharing of roads and intersections requires the Petri net generator to have a knowledge of the working environment. We currently consider robots navigating through a road network using a method of *marker-based navigation*—robots are instructed to navigate to markers rather than to absolute locations (e.g. global coordinates in the real world). Using this method, a control scheme is not limited to a single physical road layout, but has the potential to be generally applied to a number of road networks.

A simple road network is shown in Figure 5.1(a). The network is represented as a model comprising a set of four road segments and a single intersection, each of which is given a unique identifier (e.g. R1, R2, I1). Each road segment is associated with two unique markers, one at each end (e.g. R1 is associated with markers M1 and M2). Each intersection is associated with three or more markers. The markers identify the ends of road segments leading into the intersection (e.g. I1 is associated with M2, M4, M6, and M7). Markers have knowledge of their associated road segments, but not of their intersections.

Using this method of representation, each physical road network produces a unique logical environment model. The converse, however, is not true. A logical model can represent any number of physical road networks since the physical locations of the markers and the geometry of the road segments are irrelevant to the Petri net generator. The generator needs to know only the logical relationship between roads and intersections. For example, the physical road network shown in Figure 5.1(b)

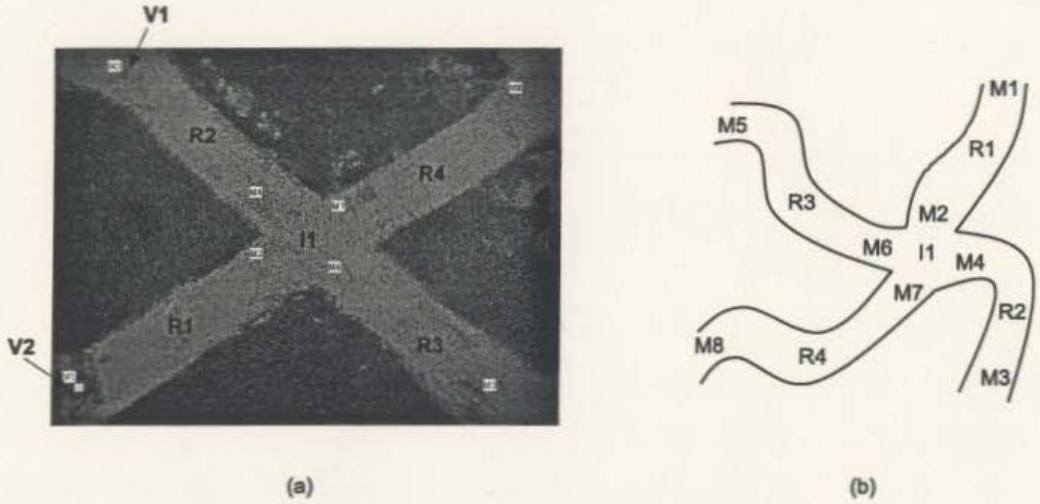


Figure 5.1: Two simple road networks with the same logical environment model.

would produce the same environment model as the road network in (a).

Currently, a fixed environment model is "hard-coded" as part of the Petri net generator. Using this method of representation, however, multiple environment models may be stored (e.g. in a database) and referred to as necessary. In this way, Petri net controllers can be generated to control multiple vehicles navigating in many environments.

5.2 High-Level Description of Robotic Tasks

The input from a dynamic scheduler to the Petri net generator is a high-level description of the tasks to be performed by each robot. A simple language for describing marker-based navigation tasks has been developed. For a single robot, a task is represented as the robot identifier followed by one or more markers, indicating that the

robot is to navigate to each marker in the order specified. In addition to the markers which delimit the ends of the road segments, two special types of markers are also used: virtual markers and "REPEAT" markers.

Virtual markers do not have any physical meaning, but are used to convey information about a robot's starting position. Because a robot may not be located at a physical marker point when a task description is sent, the virtual marker is created to identify the road segment in which the robot is initially found. REPEAT markers may be included at the end of the marker list to represent iterative behaviour which is characteristic of many robotic tasks. The REPEAT marker indicates that all the markers in the list are to be visited repeatedly in the order given.

Currently, it is assumed that road markers are listed in an order such that for each consecutive pair, both markers in the pair are either on the same road segment or are connected to the same intersection. Furthermore, in the event that a REPEAT marker is used, it is assumed that the first and last road markers in the list have been chosen so that they are at different ends of the same road segment. The REPEAT marker can then be interpreted to mean that the robot should navigate from the last marker, within a road segment to the first marker, and iterate through the marker list again.

Figure 5.1 shows two robots which must operate in the road network. An example task provided by the dynamic scheduler may require robot V1 to continuously transport ore from a muck pile at M3 to a dump site at M5. Robot V2 may be required to do the same from a muck pile at M1 to a dump site at M8. The input to the Petri net generator would be the following task description:

$$\begin{array}{l} \text{V1 MVIRTUAL1 M4 M6 M5 M6 M4 M3 REPEAT} \\ \text{V2 MVIRTUAL2 M2 M7 M8 M7 M2 M1 REPEAT} \end{array} \tag{5.1}$$

The navigation tasks for each robot are specified separately; the robots have no

knowledge of each other. It is then the responsibility of the Petri net generator to produce a central controller capable of coordinating the actions of the two robots (e.g. prevent collision at I1).

5.3 Automatic Petri Net Generation

Given the description of the environment and the high-level task descriptions for multiple robots, the Petri net generator automatically generates a Petri net controller which can coordinate the behaviours of the robots. This is accomplished in software using an application developed in Visual C++. The code that has been developed analyses the individual task descriptions, applies operating constraints, and uses a fixed set of rules to create a Petri net structure that produces coordinated behaviour. Once the structure has been determined, the software essentially mimics the actions that an operator would use to create the same Petri net manually. The concepts used in the development of the software which implements this process are described in the following sections.

5.3.1 Constraints

To generate a controller which can coordinate the behaviours of the robots, the Petri net generator must consider *constraints* imposed by the environment and the requirements for safe and efficient operation. In the current implementation, it is assumed that for safe operation,

- no more than one robot may traverse a road segment at a time
- no more than one robot may pass through a single intersection at a time.

Although initially, these constraints may appear unrealistic in some environments, it will be illustrated later (see Section 7.2.8) that with proper environment modeling,

these constraints have the potential to facilitate safe operation in a variety of road networks.

5.3.2 Resource Places

Given the above constraints, road segments and intersections are considered shared resources with mutually exclusive rules for ownership. They are represented in the Petri net with *resource places*. Resource places can be interpreted as being the "key" required for entry into a road segment or intersection. For mutually exclusive access, there is only one key per resource represented by a single token in the place. A marked resource place indicates that the resource is available; the absence of a token indicates the resource is currently "owned" by a robot. Resource places are created only for resources that are specified in the robot task descriptions. Thus, we avoid creating resource places for roads and intersections that are never used, thereby simplifying the Petri net model.

The first step in automatic Petri net generation is the creation of resource places for each of the road segments in which robots are initially found. The ID of a robot's initial road segment is determined from the robot's virtual marker. Because these road segments are currently occupied, their corresponding resources places are initially unmarked.

Once initial resource places have been created, the Petri net generator creates sub-Petri nets for each robot. The method by which these subnets are created is discussed next.

5.3.3 Sub-Petri Nets

The Petri net generator creates a sub-Petri net for each robot. The subnets are interconnected by resource places to achieve coordinated behaviour among multiple

robots. The creation of each subnet involves a simple analysis of the task description for each robot.

The task description for a robot consists of an ordered list of markers to be visited by the robot. It is assumed that the markers are listed in an order that is physically realistic. That is, both markers in each consecutive pair in the list are either on the same road segment or are connected to the same intersection. With this prerequisite, the consecutive pairs of markers can be considered in one of two categories:

1. Pairs requiring a robot to move within a road segment
2. Pairs requiring a robot to move through an intersection.

A generic Petri net structure for each category has been developed. These structures are used as basic building blocks for generating complete subnets for each robot.

Category 1: Movement Within a Road Segment

Consider the task description for robot V1 given in (5.1): "V1 MVIRTUAL1 M4 M6 M5 M6 M4 M3 REPEAT" where MVIRTUAL1 indicates that the robot's initial position is within R1. The first two markers instruct V1 to move from MVIRTUAL1 to M4, two markers belonging to the same road segment. The Petri net structure which is generated to control this movement is shown in Figure 5.2.

With the robot at MVIRTUAL1 (represented by a token in place PVIRTUAL1) and ready for its next command, transition TO M4 is enabled (Figure 5.2(a)) and fires. When the transition fires, an instruction is sent to the robot to move to marker M4 and tokens are placed in P1 and P2 (Figure 5.2(b)). The stochastic transition AT M4 is enabled by the token in P2 and begins firing. While the robot is in the process of moving to M4, P1 remains marked. When the robot reaches M4, transition AT M4 completes firing, marking P3 and hence enabling NEXT COMMAND(Figure 5.2(c)).

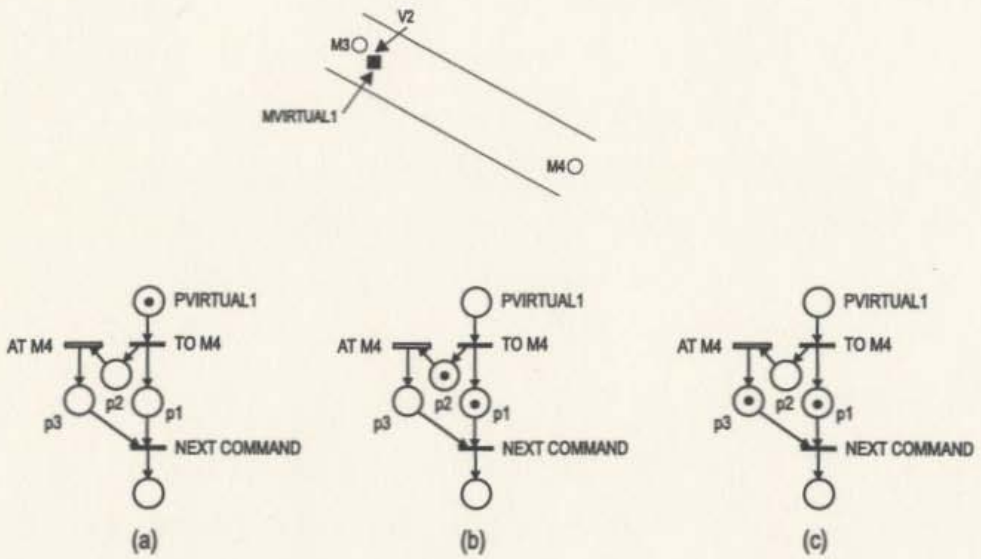


Figure 5.2: Petri net structure for Category 1: movement within a road segment.

During execution of this command, since the robot has not changed its ownership of any resources (it maintains ownership of R2), no changes to the resource places are required.

By removing the specific marker names from the transition names and replacing them with variables, the Petri net in Figure 5.2 can be transformed into a generic structure. This structure can be used to command any robot to move between any two markers on the same road segment.

Category 2: Movement Through an Intersection

Again, consider the task description for V1 given in (5.1). The second and third markers instruct V1 to move from M4 to M6 through intersection I1. Two things may happen: the robot may be permitted to proceed through the intersection, or the robot will be required to stop.

- In the first case, V1 must be able to obtain ownership of the intersection (I1) and subsequently of the road segment on the other side (R3). In the current implementation, a conservative approach is taken and V1 takes ownership of both before entering the intersection. This means that R3 is considered occupied before there is actually a robot in it. This approach, however, guarantees V1's ownership of R3 when it completes its travel through I1.

  Although this is not an issue in the current example, it becomes important in situations where multiple robots are required to share road segments. The conservative approach prevents a robot from becoming "stranded" in the intersection in the event that the destination road segment is occupied.

  As V1 enters the intersection, it must relinquish its ownership of its current road segment (R2). Similarly, when V1 reaches the other side (M6), it must relinquish its ownership of the intersection (I1).

- In the second case, if either the intersection (I1) or the destination road segment (R3) is occupied, the robot is issued a command to stop and waits until the occupied resource becomes free.

The Petri net structure used to command a robot to move through an intersection is shown in Figure 5.3. There are three resource places (shown in gray): R2, R3, and I1. The presence of a token in a resource place indicates the resource is available. Initially, the robot is at M4 (represented by tokens in places P1 and P3 corresponding to places of the same name in Figure 5.2) and the robot's current road segment R2 is not available (Figure 5.3(a)).

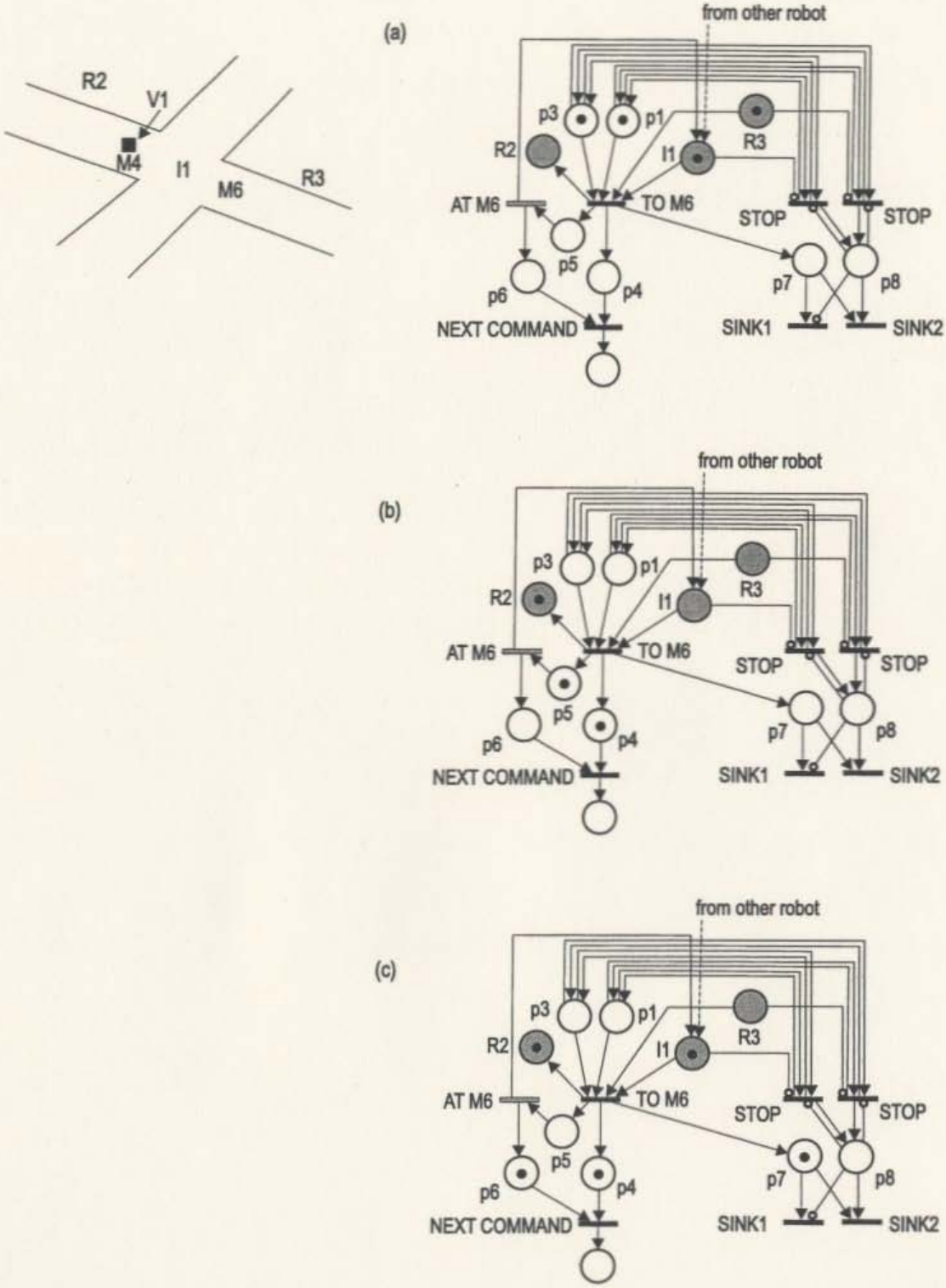
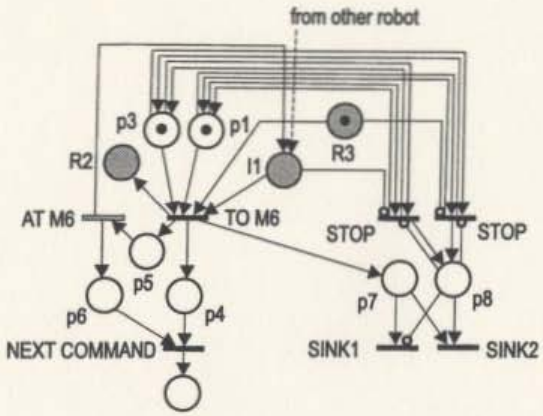
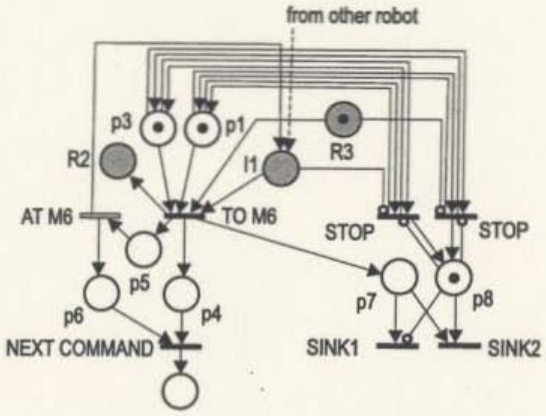


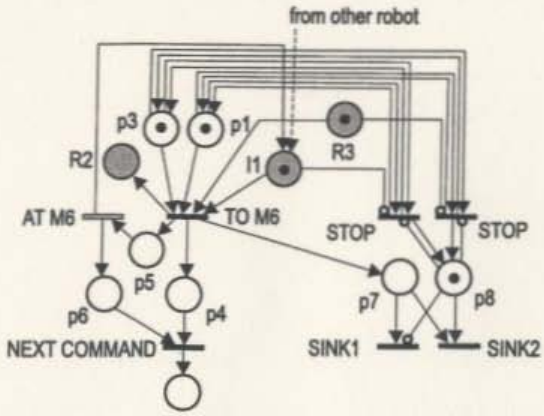
Figure 5.3: Petri net structure for Category 2: movement through an intersection.


from other robot
p3
p1
R2
I1
R3
AT M6
TO M6
STOP
STOP
p5
p7
p8
p6
p4
NEXT COMMAND
SINK1
SINK2

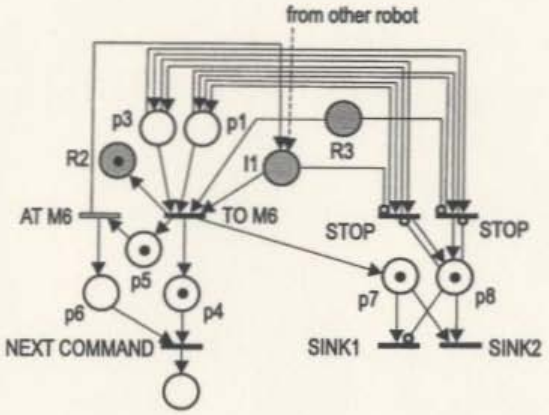
(d)


from other robot
p3
p1
R2
I1
R3
AT M6
TO M6
STOP
STOP
p5
p7
p8
p6
p4
NEXT COMMAND
SINK1
SINK2

(e)


from other robot
p3
p1
R2
I1
R3
AT M6
TO M6
STOP
STOP
p5
p7
p8
p6
p4
NEXT COMMAND
SINK1
SINK2

(f)


from other robot
p3
p1
R2
I1
R3
AT M6
TO M6
STOP
STOP
p5
p7
p8
p6
p4
NEXT COMMAND
SINK1
SINK2

(g)

- If both I1 and R3 are available, transition TO M6 is enabled and fires, and an instruction is sent to the robot to move to M6. Tokens are placed in P4 and P5, and tokens are removed from resource places I1 and R3 to indicate that the resources are in use(Figure 5.3(b)). At the same time, a token is returned to R2 since it is no longer occupied by V1. Also, an additional token is produced by the firing of TO M6 to mark P7. This token is simply consumed by the sink transition SINK1(Figure 5.3(c)). When transition AT M6 fires, a token is returned to the intersection resource place (I1).

- If either I1 or R3 is not available(Figure 5.3(d)), a STOP transition is enabled and fires, and an instruction is sent to the robot to stop moving. A token is placed in P8, and tokens are returned to P1 and P3 (Figure 5.3(e)). The resulting effect is that the robot stops and waits until both resources (I1 and R3) become free. The inhibitor arcs from P8 to the two STOP transitions prevent multiple stop commands from being issued to the robot while it is waiting. When both resources become available (Figure 5.3(f)), the TO M6 transition becomes enabled and fires as before. This time, however, the token in P8 inhibits SINK1 from firing, and instead both the token in P7 and in P8 are consumed by SINK2 (Figure 5.3(g)).

At this time, it is important to note one of the assumptions of this model. When a STOP command is issued to a robot, it is assumed that the robot receives the message and is able to stop before it enters the intersection. This means that the transition must fire before the robot actually reaches the end of the road segment leading into the intersection. The length of time required for this should consider the worst case communication delays and the dynamics of the vehicle itself. Thus, when the robot receives a stop command, it begins action at that time (e.g. deceleration) that will

result in the vehicle being stopped when it reaches the end of the road segment.

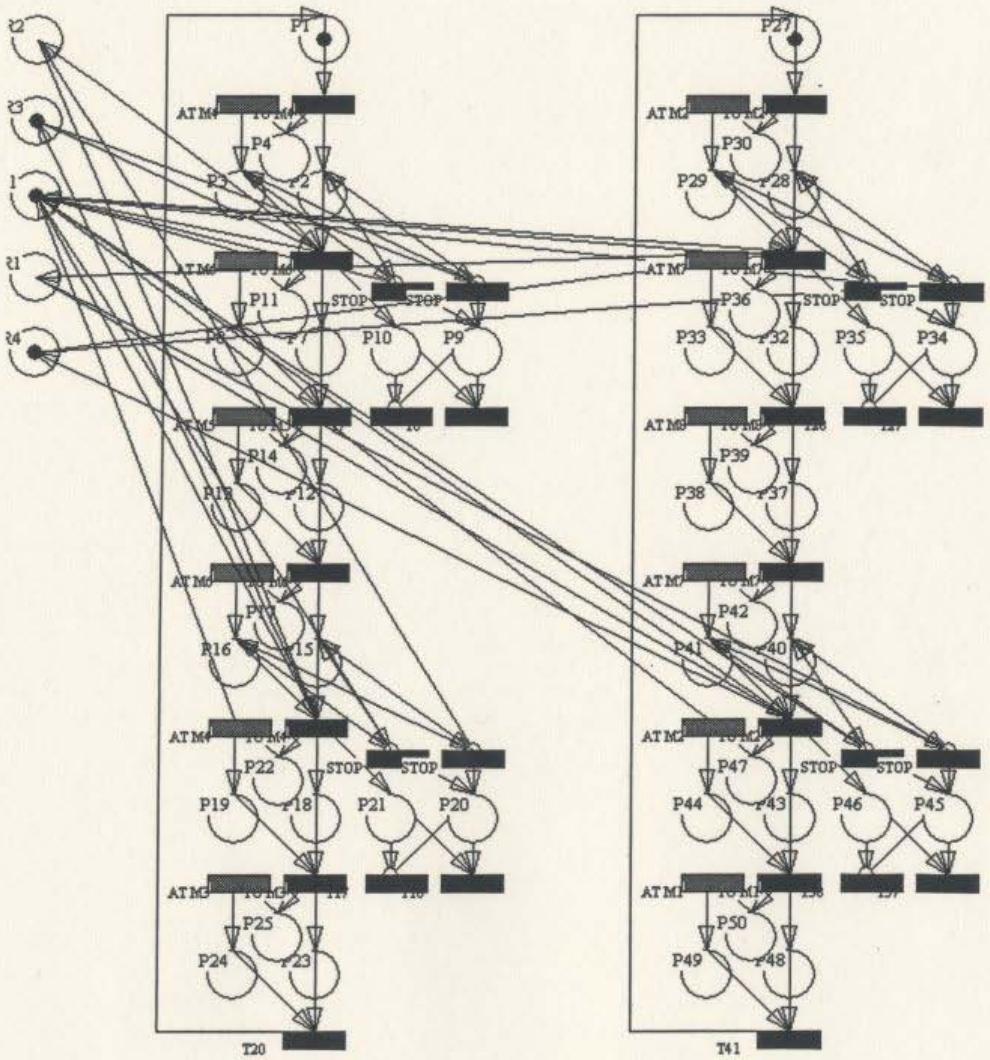
In practical applications, this can be implemented by using "pre-markers" which are located farther back from the intersection than the original markers and are customized for each type of robot. The robots could provide feedback when they reach the pre-markers, and the next command could be queued. If a robot, then, is given two consecutive MOVE commands, it may execute them both without having to slow down in between, thereby improving productivity.

As in the previous category, by removing the specific marker and resource names from the net and replacing them with variables, the Petri net in Figure 5.3 can be transformed into a generic structure. This structure can be used to command a robot to move between two markers on different sides of an intersection.

The generic structures presented for Categories 1 and 2 can be used as basic building blocks to construct a Petri net controller for a complete task. Figure 5.4 shows the Petri net that has been generated for the task described in (5.1): "V1 MVIRTUAL1 M4 M6 M5 M6 M4 M3 REPEAT V2 MVIRTUAL2 M2 M7 M8 M7 M2 M1 REPEAT".

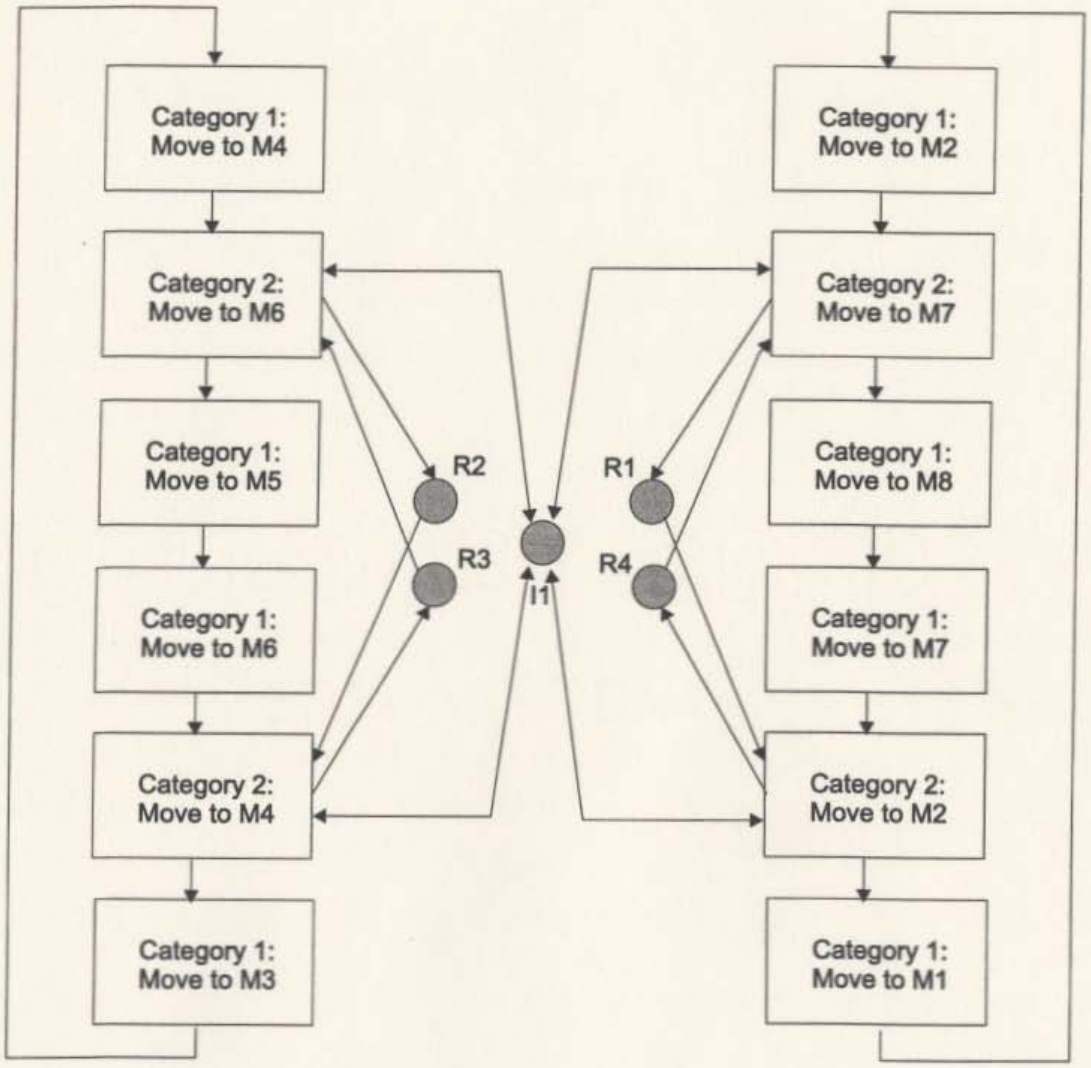
In (a), the net generated and displayed by the Petri net software application is shown. Although further work is required in the graphical layout of the Petri net, the functionality of the net is complete. Resource places are located in a column at the left side. The subnets for V1 and V2 are clearly separated, with the only interconnections being through the resource places which are used to coordinate behaviour. The arc from the bottommost transition to the topmost place in each subnet has been automatically generated in response to the REPEAT marker.

In (b), the same Petri net is illustrated as an interconnection of the basic building blocks developed for Categories 1 and 2.



(a)

Figure 5.4: Generated Petri net structure for a complete task.


Category 1:
Move to M4
Category 2:
Move to M6
Category 1:
Move to M5
Category 1:
Move to M6
Category 2:
Move to M4
Category 1:
Move to M3
R2
R3
I1
R1
R4
Category 1:
Move to M2
Category 2:
Move to M7
Category 1:
Move to M8
Category 1:
Move to M7
Category 2:
Move to M2
Category 1:
Move to M1

(b)

The Petri net structures described can accommodate an arbitrary number of mobile robots operating in any network of roads. Thus, system scalability is not limited by this method of automatic Petri net generation.

5.4 A Petri Net Interpreter

Our general framework for group robotics stresses the need for a means of managing robot interactions at execution time. Once a Petri net control structure has been automatically generated from a high-level task description, the Petri net controller is then executed to achieve *on-line control* and *runtime monitoring*. As the Petri net executes according to the rules for transition firing, instructions appropriate to the current system state are sent to individual robots. As robots complete their tasks or encounter difficulties, they provide feedback which is incorporated into the Petri net execution and used to update the state of the system.

Using our Petri net software application, it is possible to model systems using two types of transitions: *deterministic* and *stochastic*. These two types of transitions have been given different meanings in the context of on-line control and runtime monitoring.

5.4.1 Deterministic Transitions and On-Line Control

Deterministic transitions can be assigned a firing time, t, so that when the transition is enabled, it fires and does not deposit tokens in output places until after t time units have passed. Each deterministic transition can also be associated with a task-level robot command. The command is specified with a robot identifier and the task to be carried out by the robot. During Petri net execution, when a transition is fired, its associated command is sent to the appropriate robot.

For example, in Figure 5.2, transition "TO M4" has an associated robot command that instructs robot V1 to move to marker M4. When transition TO M4 fires, the instruction is sent to V1 and is intended to be carried out immediately.

5.4.2 Stochastic Transitions and Runtime Monitoring

Stochastic transitions are associated with feedback from robots. The same as deterministic transitions, stochastic transitions are enabled and fired according to the rules for Petri net execution. The duration of the firing, however, is non-deterministic. Thus, stochastic transitions are useful in modeling processes which require an unknown length of time (e.g. a robot traversing a road segment which may contain unknown obstacles). Each stochastic transition is assigned an event to monitor. Once a stochastic transition fires, it waits for the event to occur. When feedback is received indicating that the event has occurred, the transition completes its firing and deposits tokens in its output places.

In Figure 5.2, transition AT M4 is assigned to monitor the receipt of an "AT M4" message from V1. When the transition fires, it will continue to fire until appropriate feedback is received from V1. The transition then completes its firing and deposits tokens in its output places.

5.4.3 Graphical Monitoring at Runtime

As a Petri net is executed, the movement of tokens is shown as a two-dimensional animation in the Graphical User Interface of our Petri net software application. In this way, it is possible to monitor the state of the system during operation by observing the distribution of tokens presented in a graphical form.

5.5 Petri Net Analysis

One of the major strengths of the Petri net formalism is the support available for the analysis of many properties and problems associated with concurrent systems. In this section, some properties of Petri nets are defined, one particular method of analysis is explained, and finally, the current analysis capabilities of the Petri net software application that has been developed are described.

5.5.1 Petri Net Theory: Behavioural Properties

The *behavioural properties* of a Petri net are properties which depend on the initial marking of the net. Among other properties, the initial token distribution determines the *reachability*, *boundedness*, and *liveness* of a Petri net.

Reachability

When an enabled transition is fired, the marking of a Petri net is changed according to the transition firing rule. A sequence of firings will give a sequence of markings. A marking M_n is *reachable* from a marking M_o if there exists a sequence of firings that transforms M_o to M_n. The *reachability set* $R(M_o)$ of a marked Petri net is the set of all markings reachable from M_o. The *reachability problem* for Petri nets is that of determining if a marking M_n is reachable in a net (N, M_o).

For a Petri net that has been automatically generated from a task description, let M_n be a marking that represents a collision between two vehicles. Given the initial marking M_o, if it is determined that M_n is reachable from M_o, then the task can be revised to prevent the collision. If, on the other hand, it is determined that M_n is *not* reachable from M_o, then we have a formal verification that this particular collision scenario will not occur. This type of analysis can be quite valuable during the development phase of a control system.

Boundedness

A Petri net is said to be *k–bounded* if the number of tokens in each place is never greater than a finite number k for any of the reachable markings. A Petri net is said to be *safe* if it is 1–bounded: that is, the number of tokens in each place is either 1 or 0. In the context of a multiple mobile robot system, let us assume that places represent physical locations and that a token in a place represents the presence of a robot at a particular location. If the Petri net is determined to be safe, then the operation of the system is guaranteed never to attempt to force two or more robots to occupy the same space.

Liveness

A transition t in a marked Petri net is said to be *live* if, from each reachable marking, it is possible to progress through a firing sequence to another marking in which t is enabled. A marked Petri net is live if each transition is live. This means that a live Petri net guarantees deadlock-free operation. A dead Petri net is defined to be a net in which every transition is dead. Different levels of liveness have been defined, and details can be found in [30]. The liveness of a Petri net controller for a system of multiple mobile robots is important in determining the productivity of the system.

5.5.2 Petri Net Theory: Analysis Methods

A number of Petri net analysis methods exist. A method of *coverability trees* is explained next, followed by a description of the analysis module of the Petri net software application.

Coverability Trees

A Petri net with an initial marking can have as many "new" markings as there are enabled transitions. Each of these "new" markings, in turn, can generate more markings. In this way, it is possible to represent all the reachable markings as a tree, where each node represents a marking that has been generated from the initial marking and its successors, and each arc represents a transition firing that transforms one state into another. The *coverability tree* for a marked Petri net can be analyzed for a number of properties, including boundedness, safeness, dead transitions, and reachable markings.

For a bounded Petri net, the coverability tree contains all possible reachable markings, and is therefore known as the *reachability tree*, which can be used in an exhaustive method for all analysis problems. From coverability and reachability trees, corresponding coverability and reachability graphs can be drawn.

5.5.3 Software Analysis Module

Analysis capabilities within a Petri net software application may often be very useful. A simple analysis module has been developed for our Petri net software application, currently capable of performing reachability analysis on bounded nets. The implementation of this module is presented.

In performing reachability analysis, a graphical Petri net model is first translated into a mathematical representation involving an *input matrix*, an *output matrix*, and a *marking vector*. The input matrix, I, is a $p \times t$ matrix where p is the number of places in the net and t is the number of transitions. Element (i, j) of the input matrix contains the weight of the directed arc from place i to transition j. The element is set to zero when the place i is not an input place of transition j, and to -1 when the input place is connected to the transition with an inhibitor arc. Similarly, the output

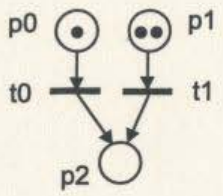
matrix, O, contains the weights of the directed arcs from transitions to places. The marking vector, M, is of length p, and contains the number of tokens in each place. The places and transitions are numbered according to the order in which they were created using the Graphical User Interface.

Transition firings are implemented by manipulating these matrices. In order to fire transition j, all the input places to transition j must contain sufficient tokens and the transition must not be inhibited. That is, for each place i where $I(i,j) = t_i$, if $t_i > 0$ and $M(i) \geq t_i$ or if $t_i = -1$ and $M(i) = 0$, then transition j will fire, removing t_i tokens from each input place i where $t_i > 0$. Furthermore, for each place k where $O(k,j) = u_k$, u_k tokens will be added to each output place k. The analysis treats all transitions as logical with zero firing time.

A recursive algorithm implementing a "depth-first" search is used to produce the coverability tree. The nodes of the tree (i.e. the markings represented by the nodes) are stored in a global set. This set is initialized to contain only one marking—the initial marking. Then for each transition that is enabled by the marking, the transition is fired, a new marking is generated and a new node added to the tree, and the reachability analysis is conducted for the new marking. The stopping condition for the recursion is set when the firing of the transition generates a marking that is already represented by a node in the tree. The arcs of the coverability tree are stored in a second set. Each time a transition is fired, a new entry in the set is created containing three fields: the current marking, the transition fired, and the new marking resulting from the firing.

The reachability analysis module also guards against state explosion for large nets. A parameter can be set to limit the maximum number of states that will be "found" by the analysis. Once this limit has been reached, the recursion is stopped, and the value returned as the "next state" following a transition firing indicates that the maximum number of states has been reached. A message is also sent to let the user

(a)



(b)

States

No.	p0	p1	p2
1	1	2	0
2	0	2	1
3	0	1	2
4	0	0	3
5	1	1	1
6	1	0	2

Transitions

Curr	Trans	Next
3	t1	4
2	t1	3
1	t0	2
5	t0	3
6	t0	4
5	t1	6
1	t1	5

(c)

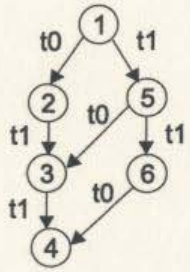


Figure 5.5: The output of the reachability analysis module for a simple Petri net.

know that a complete reachability analysis was not possible.

A simple Petri net is shown in Figure 5.5(a), and the results of the reachability analysis are shown in (b). The results are presented under two headings: "States" and "Transitions". Each row beneath the heading of "States" is a reachable marking. For example, the first row reads [1 2 0] indicating that in the initial marking, places p_0, p_1, and p_2 are marked with 1, 2, and 0 tokens respectively. There are six rows under "States"; thus this Petri net has six reachable states.

Each row under the heading of "Transitions" represents an arc in the reachability tree. The middle column shows the transition that was fired to bring the Petri net from the state number in the first column to the state number in the third column. Note that state 4 whose marking is [0 0 3] is never found in the first column under "Transitions" indicating that no transitions could be fired from state 4. That is, the

Petri net is dead. This can also be seen in Figure 5.5(c) where the textual results in (b) are represented as a graph.

The reachability tree contains information about all the reachable states of the system. The markings of a reachability tree can be subsequently analyzed to extract certain types of information which provide formal proof of the system properties. For example, consider the Petri net for a complete task shown in Figure 5.4. The tree may be modeled to show that for the subset of places that represent that a vehicle is navigating toward a marker, no more than one place is marked at a time. Thus, the controller can be guaranteed never to command a robot to move to two different markers simultaneously. This type of further analysis remains a subject of future work.

Chapter 6

Implementation and Demonstration Results

The utility of a Petri net controller for the coordination of multiple mobile robots was illustrated in a proof-of-concept demonstration. In order to demonstrate the concept of Petri net on-line control in the context of our general framework for group robotics, a replacement module for the dynamic scheduler was created and two mobile robot platforms were developed. These components of the framework were integrated into a system which was used to coordinate the actions of multiple vehicles in navigation tasks.

6.1 Task Definition Application

The role of the dynamic scheduler is to provide the Petri net generator with a description of the tasks to be performed by each robot. For the proof-of-concept demonstration, a Task Definition software application was developed to provide this functionality. Although in this application, robot scheduling is neither dynamic nor automatic, an operator is able to specify marker-based navigation tasks quickly and easily using

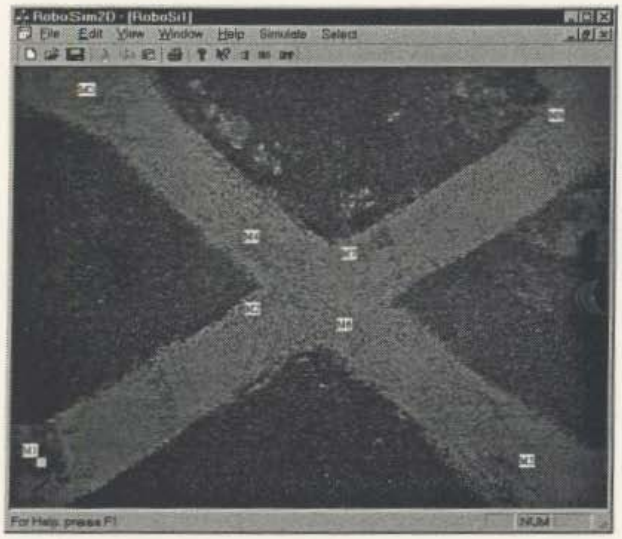


Figure 6.1: The graphical user interface for the Task Definition application.

a graphical user interface (GUI).

The GUI for the Task Definition application is shown in Figure 6.1. The interface shows an image of the road network within a scaled mining site. The locations and names of markers used to identify the ends of road segments are overlaid on the image. The initial positions of two robots inside the mine are shown as coloured squares. Through the GUI, the user is able to select a robot and then click on a series of markers to be visited by the robot in order. A "REPEAT" marker can also be added at the end to indicate iterative behaviour. In this way, high-level descriptions of navigation tasks can be created for multiple robots.

Once the navigation tasks have been specified, the task description is "sent" to the Petri net generator. The method of communication is described in Section 6.3.

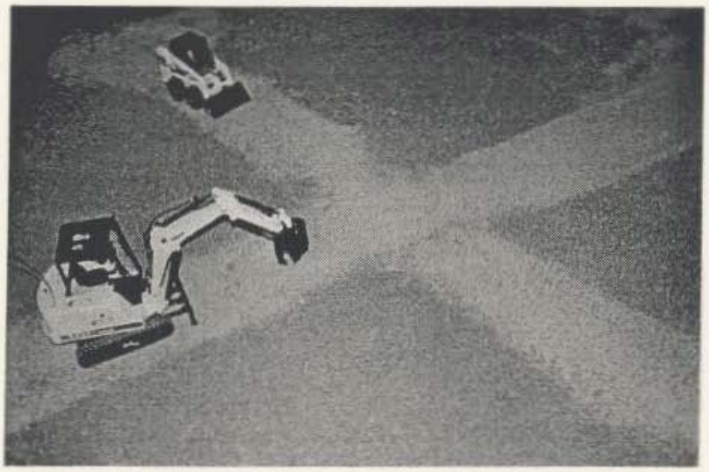


Figure 6.2: Model mining vehicles in a scaled version of a mining site.

6.2 Mobile Robot Platforms

The Petri net controller was used to control mobile robots on two different platforms: physical robots in a scaled version of a mining site, and virtual robots in an OpenGL mining environment.

6.2.1 Scaled Version of a Mining Site

A scaled version of a mining site was constructed as a test site for the Petri net on-line controller. The mining vehicles used in testing were remote-controlled models of actual construction vehicles and are shown in Figure 6.2. The remote control units for the model trucks were modified to accept commands from a PC.

A software application was developed which is able to accept high-level commands such as "Move to M1" and to send the appropriate remote-control signals to drive the truck to M1. This application requires the use of a path planner and a positioning system. For path planning, a simple straight-line method is used. For positioning,

a pan/tilt camera is mounted overhead and image processing techniques used to determine vehicle locations. In this way, a positioning system such as GPS or DGPS is emulated.

Upon task completion by a robot, the software application sends a message to the Petri net controller.

6.2.2 Virtual Mining Site

When working with physical multi-robot systems, implementation presents a number of challenges. Each robot must be outfitted with a low-level controller and a suite of sensors. Physical mobile robots can also be prone to a variety of breakdowns. Thus, for reasons including time, cost, and overall system reliability, the utility of physical multi-robot systems in testing can often be quite limited.

Virtual robots in virtual environments do not grapple with the same issues and can therefore be valuable platforms for testing. A virtual mining environment was created in OpenGL and is shown in Figure 6.3. It is possible to view any portion of the worksite from different viewing angles. Multiple vehicles are easily placed in the environment with keyboard commands. As with the physical robots, the virtual robots are able to execute high-level commands such as "Move to M1" using a simple straight-line path planner. When the task is complete, a message is again sent to the Petri net controller.

It is intended that the interface between the Petri net controller and the virtual robots will be identical to the interface to the physical robots. In this way, physical and virtual robots can be easily interchanged in a manner that is transparent to the Petri net controller. The implication is that control schemes may be tested using virtual robots and subsequently used to control physical robots to achieve the same

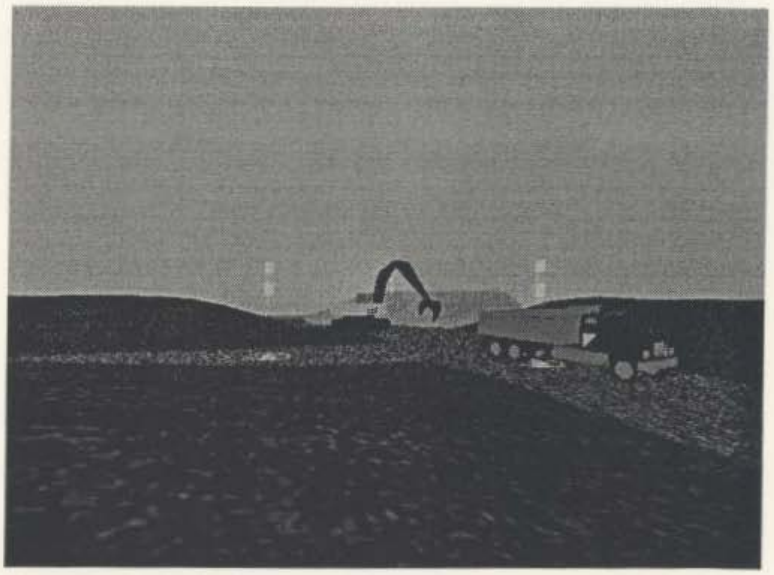


Figure 6.3: Mining vehicles in a virtual environment created in OpenGL.

system task, with few if any modifications. Furthermore, it would be possible to have physical robots interact with virtual robots in a coordinated fashion.

6.3 Communications

Once a task description has been created using the Task Definition Interface, the description is sent to the Petri net generator to be translated into a Petri net structure. Also, during execution, the Petri net controller sends commands to the robots and receives feedback from the robots when tasks are completed. Thus, a means of communication among the components of the framework is required.

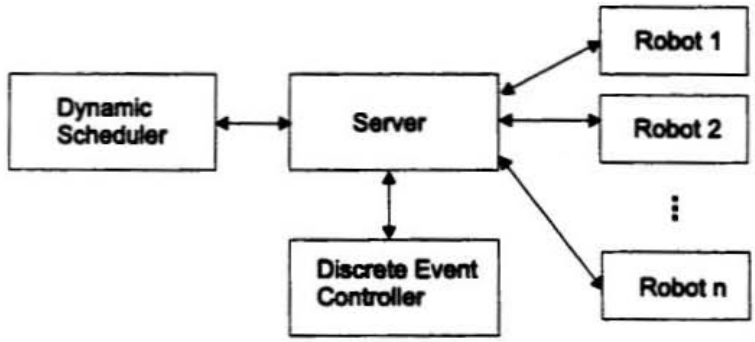


Figure 6.4: Server-Client architecture used for communication between components.

6.3.1 Windows Sockets

The components of the framework communicate over a local area network using Transmission Control Protocol/Internet Protocol (TCP/IP). Each software application associated with a component includes a Windows Socket. A socket is a communication endpoint, an object through which an application sends or receives data across a network. Sockets are bidirectional, and can therefore both send and receive messages.

6.3.2 Server-Client Architecture

In addition to the software applications associated with the components of the framework, a separate application was developed as a Central Server for the communications architecture. Each component of the framework must connect as a client to the central server. This architecture is shown in Figure 6.4.

The components communicate by message-passing. Each message contains information identifying the intended recipient of the message. All messages are sent to the server and are subsequently broadcast to all the clients. The components recognize and process messages intended for them, and ignore all the other messages.

Through this method of communication, there is no limit to the number of components that can be added to the system. The system is scalable since more robots may be added without significant changes to the other components in the architecture. Furthermore, since the components communicate through a network, this implementation would allow an operator to control robotic tasks at a remote location.

6.4 Demonstration Results

A proof-of-concept demonstration was designed and implemented to illustrate the utility of Petri net control in the context of a general framework for group robotics. A Petri net controller was used successfully to provide coordinated navigation of two vehicles in a simple road network.

The task description required two robots to navigate repeatedly along intersecting paths. The task description was as in (5.1):

V1 MVIRTUAL1 M4 M6 M5 M6 M4 M3 REPEAT

V2 MVIRTUAL2 M2 M7 M8 M7 M2 M1 REPEAT

Figure 6.5 illustrates the progress of the task at different stages and their corresponding Petri net states (only a partial Petri net is shown). The Petri net controller guarantees coordinated behaviour between the two vehicles.

(a) The robots are at their starting positions. Two transitions are enabled: for V1, "TO M4" and for V2, "TO M2".

(b) V2 completes its task before V1, and is given permission to enter the intersection. The transition "TO M7" is enabled and fires, and V2 begins moving through the intersection.

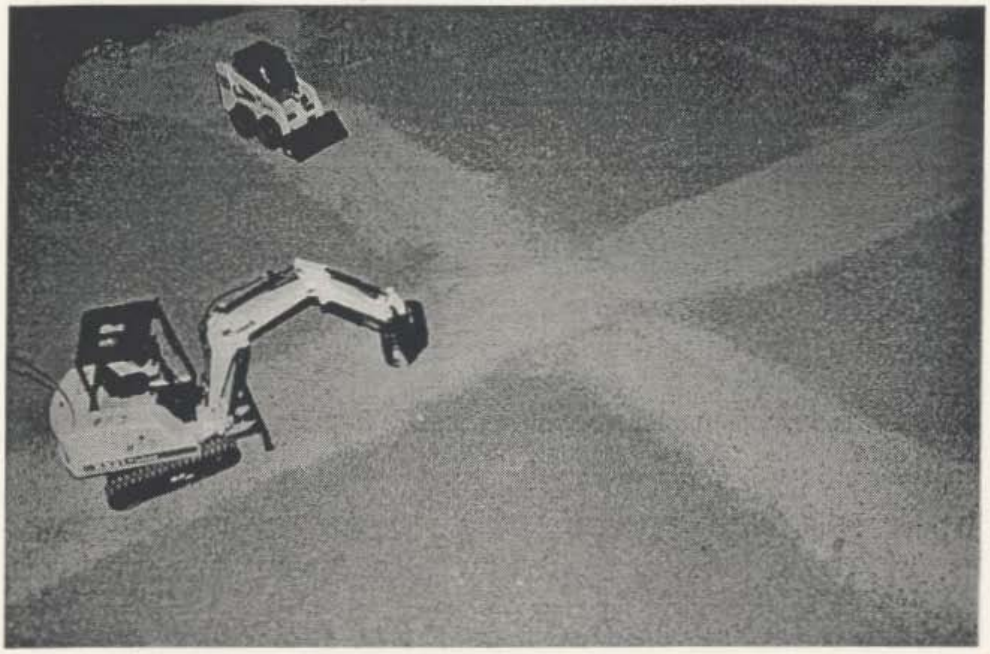
(c) In the meantime, V1 arrives at M4, but the absence of a token in I1 "un-inhibits" the "STOP" transition. V1 waits at the intersection.

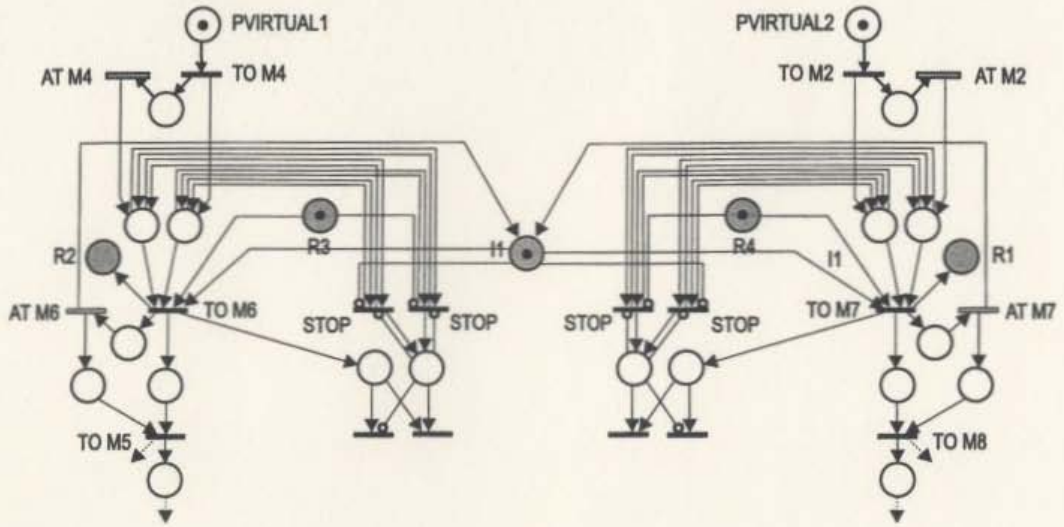
(d) V2 finishes crossing the intersection and begins moving to M8. A token is returned to place I1 which then enables the "TO M6" transition of V1.

(e) V1 crosses the intersection, returns a token to I1, and moves to M5.

The same task was demonstrated using two virtual vehicles in the virtual environment. The final experiment combined the two and demcnstrated a physical robot in coordinated behaviour with a virtual robot. Thus, from the perspective of the Petri net controller, the type of vehicle being controlled is transparent.

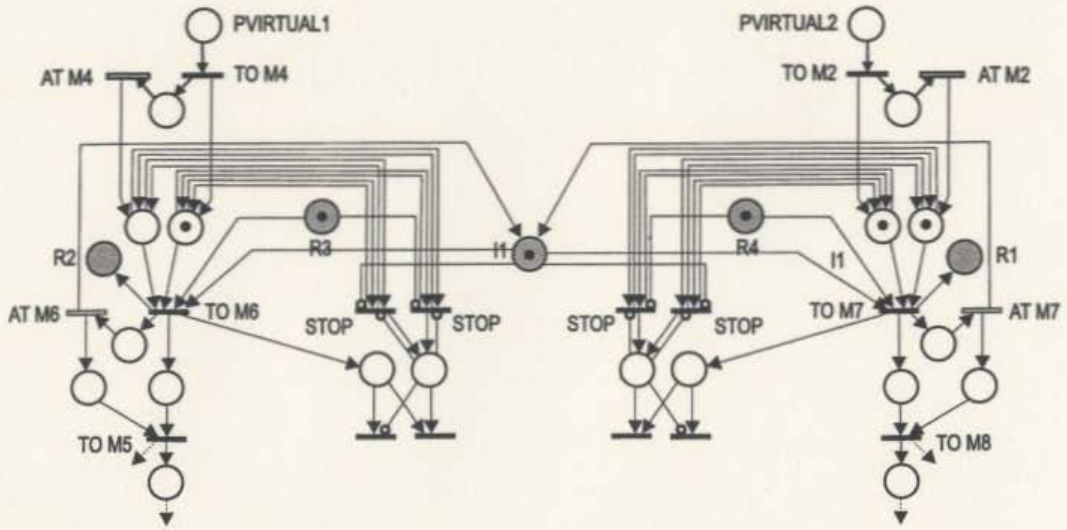
Although the demonstrations involved only two vehicles in a simple road network with a single intersection, the demonstration is easily scalable. Because the rules for automatic Petri net generations are generic, a controller can be created for an arbitrary road network given that it is represented in the proper format. The rules for coordinated navigation are also scalable; the number of vehicles is easily increased without any changes to the method by which the controller is generated.



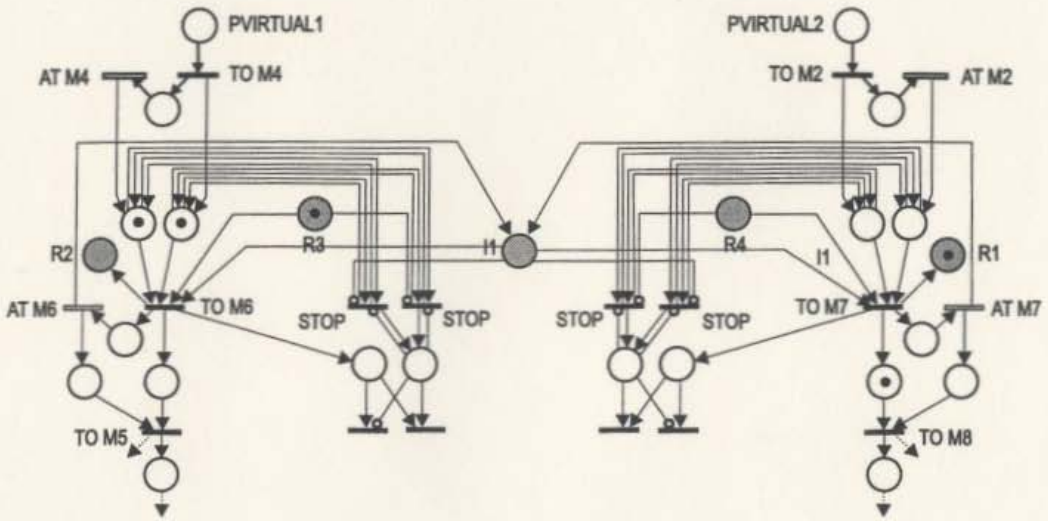


(a)

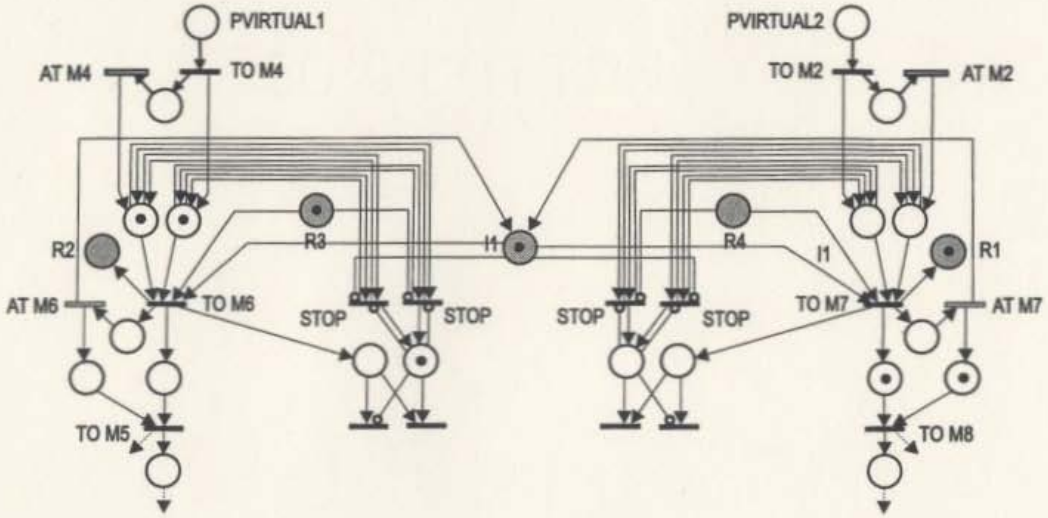
Figure 6.5: Stages of the demonstration task and their corresponding Petri net states.


PVIRTUAL1
AT M4
TO M4
PVIRTUAL2
TO M2
AT M2
R2
R3
I1
R4
I1
R1
AT M6
TO M6
STOP
STOP
STOP
STOP
TO M7
AT M7
TO M5
TO M8

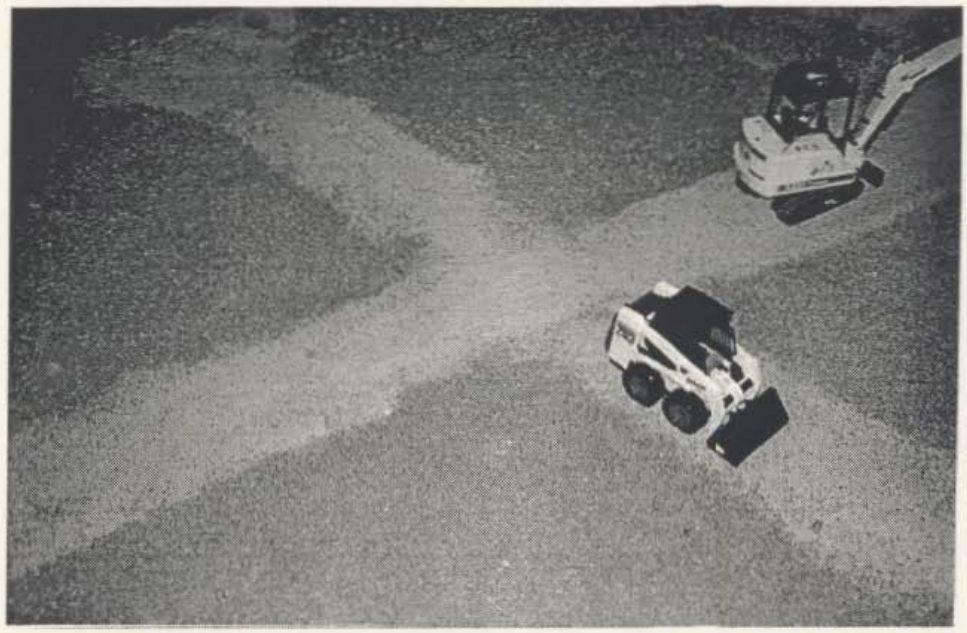
(b)

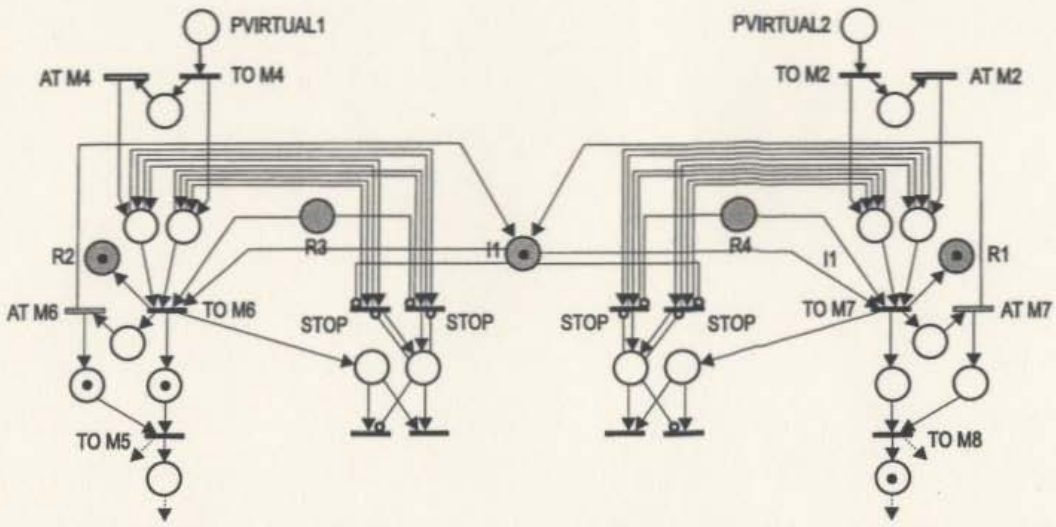

PVIRTUAL1
AT M4
TO M4
PVIRTUAL2
TO M2
AT M2
R2
R3
I1
R4
I1
R1
AT M6
TO M6
STOP
STOP
STOP
STOP
TO M7
AT M7
TO M5
TO M8

(c)


PVIRTUAL1
AT M4
TO M4
PVIRTUAL2
TO M2
AT M2
R2
R3
I1
R4
I1
R1
AT M6
TO M6
STOP
STOP
STOP
STOP
TO M7
AT M7
TO M5
TO M8

(d)




PVIRTUAL1
AT M4
TO M4
PVIRTUAL2
TO M2
AT M2
R2
R3
I1
R4
I1
R1
AT M6
TO M6
STOP
STOP
STOP
STOP
TO M7
AT M7
TO M5
TO M8

(e)

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In applications where groups of mobile robots are required to operate in semi-structured environments, there is a need to coordinate robot behaviours and to accommodate changes in a dynamic environment. In response to this need, a general framework for group robotics has been developed. Within this framework, a discrete event controller is used for on-line control and runtime monitoring.

Research in the development of a Petri net on-line controller has been described. From a high-level task description, a set of rules have been used to automatically generate a Petri net structure that provides coordinated behaviour. The Petri net can then be executed to send instructions to robots and to incorporate feedback from the robots at runtime. This method of automatic Petri net generation and Petri net interpretation has been used to control mobile robots in a proof-of-concept demonstration. In a laboratory setting, the Petri net controller was able to coordinate the behaviour of two robots in marker-based navigation tasks.

The work completed to date has provided insight into the utility of a Petri net

formalism in the control of multiple mobile robotic systems requiring coordinated behaviour. The concept of Petri net on-line control shows significant promise as a means of developing systems capable of addressing the dynamic nature of robot teams and their operating environments. In particular, the automatic generation of a Petri net control structure from a high-level task description appears to be a key factor in system reconfigurability, and automatic Petri net generation in this context appears to be novel with respect to work described in the literature.

The results of this research have been very encouraging. It is apparent, however, that many research challenges remain before a system can be implemented for use in industrial applications. The focus of the future work needs to be in further development of a discrete event controller, as well as in the development of other components of our general framework for group robotics. This work in the area of Petri net on-line control, however, has provided a good starting point.

7.2 Future Work

The work completed to date has provided a great deal of insight into the potential for using Petri nets to control cooperative mobile robotic tasks. The research that has been conducted has been valuable in illustrating the utility of Petri net control in a proof-of-concept demonstration. A number of research challenges, however, remain to be investigated in the future.

7.2.1 Hierarchical Modeling

For complex systems, Petri net models of tasks can quickly grow in size and complexity. Methods of hierarchical decomposition based on Petri nets have been developed which may make the complexity of such systems more manageable[12]. A Petri net

is used to describe a task at a high level. The Petri net is then decomposed in a stepwise manner into lower-level Petri nets in which transitions can be either directly implemented by control commands or command sequences, or further decomposed into lower level nets. The method of hierarchical decomposition allows a Petri net to retain some important properties.

The current implementation of a Petri net based on-line controller is capable of generating high-level nets intended to provide coordination between multiple robots. The method of automatic generation assumes that high-level subtasks (e.g. navigating from point A to B) are able to be completed by the mobile robots.

In order to facilitate successful completion of the subtasks, on-board controllers are required at the robot level. Lower-level Petri net controllers, derived from a systematic decomposition of the high-level controller, could be used. In this way, the properties of the high level net can be preserved. On-board Petri net controllers may also be an effective means of modeling subtasks to incorporate task-preserving human intervention. The subtask can be modeled as a sequence of discrete events in a way that will allow seamless transfer of control between human and machine.

Further research is required in methods of hierarchical decomposition of high level Petri nets for coordinated control of multiple mobile robots. Also, the modeling of robotic subtasks in a way that will allow task-preserving human intervention remains to be investigated.

7.2.2 Coloured Petri Nets

In *coloured Petri nets*[23], tokens are given attributes called *colours*. Transitions can have different firings which depend on the colours and numbers of tokens in the input places to the transition. Using the theory of coloured Petri nets, it may be possible to "fold" identical parts of an ordinary Petri net into a single coloured Petri net[43],

thereby simplifying the Petri net structure without losing any modeling capabilities. The original set of places is partitioned into a set of disjoint classes, and each class of places is replaced by a single place. The colour of each token indicates which of the original places the token belongs to. Similarly, the set of transitions is partitioned into a set of disjoint classes, and each class is replaced by a single transition with different firings to represent the original transitions.

Mobile robotic systems often comprise multiple robots operating concurrently while performing similar tasks. The modeling approach taken in this work generates a separate subnet for each vehicle. It may be possible to "fold" each of these subnets into a single coloured Petri net. For complex systems, the gain in simplifying the visual representation of the Petri net may be significant.

Techniques for modeling coordinated mobile robot tasks using coloured Petri nets remain to be developed. As well, extensions to our Petri net software application are required to accommodate the full functionality required by simulations "in colour".

7.2.3 Petri Nets and Time

In order to study performance aspects of a multiple robot system, the duration of various robotic tasks must be taken into account. For example, it may be desirable to monitor the average waiting time of robots at intersections as a means of determining the optimality of the system. Although the current Petri net software application allows a firing time to be assigned to a deterministic transition, the simulation capabilities involving time remain limited. Future work could include developing the application further to consider time.

In particular, two areas of development are recommended. Firstly, the Petri net application could allow a minimum and maximum firing time to be assigned to a transition. For robotic applications in semi-structured environments, the nondeter-

ministic nature of tasks makes it nearly impossible to specify a fixed task duration. On the other hand, it is reasonable to expect that a task will be completed within a particular time interval. Task durations outside of this interval could signal that a problem has occurred and that human intervention may be required. Thus, rather than assigning a fixed firing time to a transition, it would be useful to be able to associate a "window of time" with the transition.

Secondly, the analysis capabilities of the Petri net software application should also be extended to include time. In this way, it will be possible to evaluate certain performance characteristics of the system such as average robot idle time and system throughput. Also, in determining whether or not a forbidden state will occur, although a logical Petri net analysis without time may indicate that the forbidden state is indeed reachable, timed analysis may in fact reveal that the forbidden state will never occur. That is, some of the logical states of the Petri net may be masked by timing effects. This type of analysis could be useful in developing controllers that are less conservative.

7.2.4 Synthesis Techniques

A critical component of our general framework for group robotics is the automatic Petri net generator. Given a high-level task description for multiple mobile robots, how do we build a controller that (a) achieves the task described, and (b) preserves certain properties (e.g. absence of deadlock)?

Some simple rules for automatically generating a Petri net have been presented. These rules produce nets that coordinate navigation tasks while ensuring proper resource sharing. Although they have been very useful in providing insight into the role of automatic Petri net generation, these rules have many limitations. They are only useful for navigation tasks. In accommodating the sharing of roads and intersections,

the rules are overly conservative. For example, before a vehicle is permitted to enter an intersection, not only must the intersection be free, but the road segment to be entered at the other side of the intersection must also be free.

Further research into formal Petri net synthesis techniques could be very beneficial in overcoming these limitations. A number of Petri net synthesis techniques exist [6, 29] which provide methods of building Petri nets to meet certain constraints. By building a net to meet the constraints, the resulting Petri net is *guaranteed* to have desired properties. Use of these formal techniques and algorithms would potentially allow the synthesis of Petri net controllers which can accommodate complex tasks.

7.2.5 Analysis Techniques

One of the greatest strengths of a Petri net based formalism is the support available for the analysis of many properties and problems associated with concurrent systems. A complete Petri net software package should include a variety of analysis capabilities.

The current Petri net software application has limited capabilities for analysis; a reachability tree can be produced for a bounded net. For complex unbounded Petri nets, reachability trees are not always practical due to a state explosion problem. Furthermore, it is often the case that all the detailed results of reachability analysis are not required. Thus, the current Petri net software application would benefit from the development of more options for analysis. For example, *structural analysis*, analysis based on the structure of a Petri net, can often be useful. In particular, *invariant analysis* seems to be a popular approach. More details of invariant analysis can be found in [30].

In the context of a general framework for group robotics, it is possible that the role of Petri net analysis may be redundant if sophisticated synthesis techniques are used. If it is possible from a given task description to synthesize a net that is guaranteed to

have certain properties (e.g. absence of deadlock and collision), there may no longer be a need to analyze the net for these properties. In this case, it may be more useful to analyze the task description for inherent collision- and deadlock-causing instructions. If, from the task description, it is *not* possible to synthesize a net that is guaranteed to have desired properties, the task description should be revised. The role of an analysis module in the context of the framework is an area that requires further consideration.

7.2.6 Task Specification Language

In the context of our general framework for group robotics, the Petri net generator requires a detailed specification of a robotic task. As discussed in Section 7.2.5, it may be desirable to analyze the task description for certain properties. If the nature of the task description is such that the resulting system will result in collision and/or deadlock, the task can be revised.

In order to perform an analysis at this level, a formal task specification language is required. The task specification language should allow task descriptions to be formally analyzed for certain properties. At present, the language used in task specification is relatively simple, consisting of an ordered list of markers for each robot, and is limited to the specification of marker-based navigation tasks. Realistic tasks involving multiple robots, however, are more complex. The task specification language should be able to describe non-navigation tasks, for example, digging, dumping, drilling, and blasting for a mining application. In addition, the specification language should be capable of describing tasks which must be performed in a particular sequence (e.g. robot 1 must blast before robot 2 can begin hauling). The language may accommodate a number of flow–control structures such as "if–then" and "do–while" statements.

Furthermore, it is paramount that an operator-friendly interface be developed which can be used in task specification. In general, the operator in charge of specifying

tasks will not be formally trained in describing tasks using the specification language. Thus, a friendly interface is required to hide the details of the language from the operator.

7.2.7 Petri Nets in Mathematical Form

Petri Net Theory: Mathematical Representations

A Petri net and its dynamic behaviour can be described and analyzed mathematically. Matrix equations have been presented which govern the behaviour of concurrent systems modeled by Petri nets[30].

Incidence Matrix: For a Petri net N with n transitions and m places, the incidence matrix $A = [a_{ij}]$ is an $n \times m$ matrix of integers where

$$a_{ij} = a_{ij}^{+} - a_{ij}^{-}$$

and a_{ij}^{+} is the weight of the arc from transition i to output place j and a_{ij}^{-} is the weight of the arc from transition i to its input place j. That is, a_{ij}^{-}, a_{ij}^{+} and a_{ij} respectively represent the number of tokens removed, added, and changed in place j when transition i fires once. For a Petri net with marking M, transition i is enabled if and only if

$$a_{ij}^{-} \leq M(j) \quad j = 1, 2...., m$$

State Equation: The marking of a Petri net M_k is written as an $m \times 1$ column vector where the j^{th} entry of M_k represents the number of tokens in place j after the k^{th} firing in a firing sequence. The k^{th} firing vector u_k is an $n \times 1$ column vector with only one nonzero entry, a 1 in the i^{th} position indicating that transition i fires at the k^{th} firing. The state equation for a Petri net can then be written

$$M_k = M_{k-1} + A^T u_k \quad k = 1, 2, \ldots$$

Example: For the simple Petri net shown in Figure 5.5(a), the initial marking $M_o = [\ 1\ \ 2\ \ 0\ ]^T$ is changed to marking $M_1 = [\ 0\ \ 2\ \ 1\ ]^T$ by the firing of t_0 (represented by the first element in the firing vector). The state equation is as follows:

$$\begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 0 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

In addition to describing the behaviour of the Petri net, these matrices can be used in mathematical analysis (e.g. reachability and invariant analysis).

Petri Net Representations in Software

Currently, our Petri net software application allows Petri nets to be created graphically. From this graphical representation, a mathematical representation is derived and used to interpret the net.

The concept of starting with a graphical Petri net and translating it into mathematical form for execution can be valuable when nets are manually created using a GUI. In the context of automatic Petri net generation, however, this approach has some limitations. Given the current implementation of the Petri net software application, the automatic Petri net generator must create a graphical Petri net, described in terms of the on-screen placement of places and transitions. However, since it can be a nontrivial task to determine the "best" graphical layout for a net, the controllers that can be generated automatically are severely limited by an ability to represent them graphically.

The reverse strategy is therefore proposed in our general framework for group robotics. The output of the Petri net generator is a mathematical net description. From this, a Net Visualizer component produces a graphical representation.

The Petri net generator needs to be modified to automatically generate nets in

mathematical form rather than in graphical form. A separate visualization module needs to be developed to implement algorithms for translating nets from matrix representation to graphical representation. Although not required for Petri net interpretation, the visualization module should not be eliminated since one of the strengths of Petri net theory is that the graphical representation of a net greatly enhances the ability to monitor the state of a system with ease.

7.2.8 Environment Modelling

The rules for automatic Petri net generation have been developed based on the constraint that for safe operation, no more than one robot may traverse a road segment at a time, and no more than one robot may pass through a single intersection at a time. Although initially, these constraints may appear unrealistic in some environments, with proper environment modeling, they have the potential to facilitate safe operation in road networks with less restrictive functional requirements. As illustrative examples, two possible environment models are presented, one which allows multiple vehicles to travel (in the same direction) in a road segment and another which allows vehicles to travel both ways in a road segment.

7.2.9 Multiple vehicles in a road segment

To accommodate the travel of multiple vehicles in the same direction within a road segment, a road segment can be modeled as a series of "road sub-segments" which may be traversed by only one robot at a time. Thus, rather than a single resource place for each large road segment, a resource place can be generated for each sub-segment. If the sub-segments are connected by "dummy intersections" (intersections requiring zero time to cross), no changes to the rules for Petri net generation are required. Figure 7.1 shows a modified model for road segment R1 that will allow

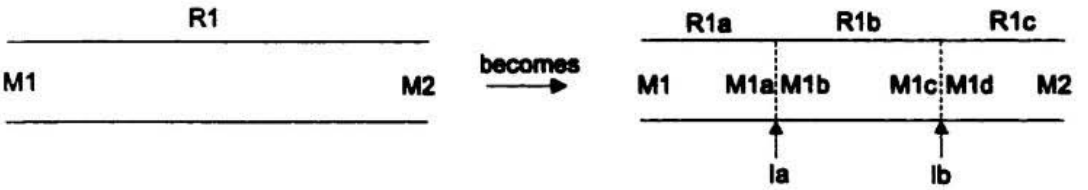


Figure 7.1: A road model to allow three robots to travel within R1 simultaneously.

Figure 7.2: A road model that will allow two-way travel in a road segment.

three robots to travel within R1 simultaneously.

7.2.10 Two-way navigation within a road segment

The current environment model allows navigation through a road segment in only one direction at a time. In many applications, it is highly likely that two-way traffic will be required in road segments. In this case, a segment can be modeled as two "side segments" (Figure 7.2). If it is assumed that the task description for each robot obeys the directions of travel for each side segment (i.e. no robot is instructed to travel the "wrong" way in a road), the current rules for Petri net generation need not be modified.

It is recommended that the modeling of environments be further investigated. Improved modeling methods and techniques for Petri net synthesis will potentially

allow controllers to be synthesized for robots operating in a variety of environments.

7.2.11 Facilitating Dynamic Re-scheduling

In the context of our general framework for group robotics, the Petri net generator is intended to receive from a dynamic scheduler a description of tasks for individual robots. The individual robotic task descriptions are then merged into a centralized control scheme which can be formally verified to ensure coordinated behaviour among the multiple robots.

In the event that the operating conditions of the system change in such a way that the behaviour of the group of robots is affected, the dynamic scheduler reassigns the tasks to the robots, and a new Petri net controller is created.

From the perspective of the discrete event controller, the transition from one control scheme to a "rescheduled" control scheme remains an area for future work. The Petri net generator is not currently equipped to communicate with the dynamic scheduler in this iterative fashion, and further development to accommodate this feature is required. Also, it is quite likely that some robots will require rescheduling when other robots are in the process of completing a task. Further investigation is required to determine an appropriate way to change control schemes in mid-task.

References

[1] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi Robot Cooperation in the Martha Project. *IEEE Robotics and Automation Magazine*, 5(1), pp. 36–47, 1998.

[2] R. Alami, F. Ingrand, and S. Qutub. A Scheme for Coordinating Multi-Robot Planning Activities and Plans Execution. *Proceedings of the ECAI '98*, Brighton, UK, LAAS Report No. 98174, 1998.

[3] D. Andreu, J. Pascal, and R. Valette. Fuzzy Petri Net-Based Programmable Logic Controller. *IEEE Transactions on Systems, Man, and Cybernetics — Part B: Cybernetics*, Vol. 27, No. 6, pp. 952–961, December 1997.

[4] R. C. Arkin and T. Balch. Cooperative Multiagent Robotic Systems. Chapter in *Artificial Intelligence and Mobile Robots*, D. Kortenkamp, R. P. Bonasso, and R. Murphy (Eds.), MIT/AAAI Press, 1998.

[5] K. Azarm and G. Schmidt. A Decentralized Approach for the Conflict-Free Motion of Multiple Mobile Robots. *Advanced Robotics*, Vol. 11, No. 4, pp. 323–340, 1997.

[6] Z. Banaszak and M. H. Abdul-Hussin. Algorithm of Live and Conflict-Free Petri Nets Synthesis for Prescribed System Performance. *Engineering and Technology*, Vol. 17, No. 2, pp. 154-173, 1998.

[7] N. G. Bourbakis. A Traffic Priority Language for Collision-Free Navigation of Autonomous Mobile Robots in Dynamic Environments. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, Vol. 27, No. 4, pp. 573–587, August 1997.

[8] B. L. Brumitt and A. Stentz. GRAMMPS: A Generalized Mission Planner for Multiple Mobile Robots in Unstructured Environments. *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp. 1895–1902, May 1998.

[9] A. Caloini, G. Magnani, and M. Pezze. A Technique for Designing Robotic Control Systems Based on Petri Nets. *IEEE Transactions on Control Systems Technology*, Vol. 6, No. 1, pp. 72–87, January 1998.

[10] P. Caloud, W. Choi, J. Latombe, C. Le Pape, and M. Yim. Indoor Automation with Many Mobile Robots. *Proceedings of the 1990 IEEE International Workshop on Intelligent Robots and Systems*, Japan, pp. 67–72, July 1990.

[11] Y. U. Cao, A. S. Fukunaga, and A. B. Kahng. Cooperative Mobile Robotics: Antecedents and Directions. *Autonomous Robots*, 4, pp. 1–23, 1997.

[12] T. Cao and A. C. Sanderson. Task Decomposition and Analysis of Robotic Assembly Task Plans Using Petri Nets. *IEEE Transactions in Industrial Electronics*, Vol. 41, No. 6, pp. 620–630, December 1994.

[13] O. Causse and H. I. Christensen. Hierarchical Control Design Based on Petri Net Modeling for an Autonomous Mobile Robot. *Intelligent Autonomous Systems*, U. Rembold et al. (Eds.), pp. 665–678, IOS Press, 1995.

[14] O. Causse and L. H. Pampagnin. Management of a Multi-Robot System in a Public Environment. *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vol. 2, pp. 246–252, 1995.

[15] D. Crockett, A. Desrochers, F. DiCesare, and T. Ward. Implementation of a Petri Net Controller for a Machining Workstation. *Proceedings of the 1987 IEEE International Conference on Robotics and Automation*, Raleigh, North Carolina, pp. 1861–67, March 1987.

[16] P. Freedman. Time, Petri Nets, and Robotics. *IEEE Transactions on Robotics and Automation*, Vol. 7, No. 4, pp. 417–433, August 1991.

[17] E. Freund and J. Rossman. Projective Virtual Reality: Bridging the Gap Between Virtual Reality and Robotics. *IEEE Transactions on Robotics and Automation*, Vol. 15, No. 3, pp. 411–422, June 1999.

[18] D. W. Gage. Development and Command-Control Tools for Many-Robot Systems. *Proceedings of SPIE - The International Society for Optical Engineering*, Philadelphia, PA, USA, pp. 121–129, October 1995.

[19] R. Gosine, R. Hale, F. Hwang, J. King, M. Rokonuzzaman, and J. Seshadri. A General Framework for Group Robotics with Applications in Mining. *International Advanced Robotics Program, First International Workshop on Advances in Robotics for Mining and Underground Applications*, Brisbane, Australia, October 2000 (submitted).

[20] R. Gosine, M. Rokonuzzaman, R. Hale, F. Hwang, J. King, , and J. Seshadri. An Approach to Multi-Robot Cooperation Under Human Supervision. *Ocean Engineering Handbook*, CRC Press, 2000 (in press).

[21] M. Hassoun and C. Laugier. Towards a Real-Time Architecture to Control an Autonomous Vehicle in Multi-Vehicle Environment. *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp. 1134–1140, July 1993.

[22] J. Jang, P. Koo, and S. Y. Nof. Application of Design and Control Tools in a Multirobot Cell. *Computers and Industrial Engineering*, Vol. 32, No. 1, pp. 89–100, 1997.

[23] K. Jensen. *Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use*, Volumes 1–3, New York: Springer-Verlag, 1997.

[24] P. Lima, H. Gracio, V. Veiga, and A. Karlsson. Petri Nets for Modeling and Coordination of Robotic Tasks. *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, San Diego, California, pp. 190–195, October 1998.

[25] D. C. MacKenzie, R. C. Arkin, and J. M. Cameron. Multiagent Mission Specification and Execution. *Autonomous Robots*, 4, pp. 29–52, 1997.

[26] S. Mascaro and H. Asada. Hand-in-Glove Human-Machine Interface and Interactive Control: Task Process Modeling Using Dual Petri Nets. *Proceedings of the 1998 IEEE International Conference on Robotics and Automation*, Leuven, Belgium, pp. 1289–1295, May 1998.

[27] B. J. McCarragher and H. Asada. The Discrete Event Control of Robotic Assembly Tasks. *Transactions of the ASME*, Vol. 117, pp. 384–393, September 1995.

[28] L. Montano, F. J. Garcia, and J. L. Villarroel. Using the Time Petri Net Formalism for Specification, Validation, and Code Generation in Robot-Control

Applications. *The International Journal of Robotics Research*, Vol. 19, No. 1, pp. 59–76, January 2000.

[29] D. J. Mu and F. DiCesare. A Review of Synthesis Techniques for Petri Nets. *Proceedings of Rensselaer's Second International Conference on Computer Integrated Manufacturing*, Los Alamitos, CA, USA, pp. 348–355, 1990.

[30] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE*, 77(4), pp. 541–579, 1989.

[31] T. Murata, N. Komoda, K. Matsumoto, and K. Haruna. A Petri Net-Based Controller for Flexible and Maintainable Sequence Control and its Applications in Factory Automation. *IEEE Transactions on Industrial Electronics*, Vol. IE-33, No. 1, pp. 1–8, February 1986.

[32] F. R. Noreils. Toward a Robot Architecture Integrating Cooperation Between Mobile Robots: Application to Indoor Environment. *The International Journal of Robotics Research*, Vol. 12, No. 1, pp. 79–98, February 1993.

[33] P. Oliveira, A. Pascoal, V. Silva, and C. Silvestre. Mission Control of the MARIUS Autonomous Underwater Vehicle: System Design, Implementation, and Sea Trials. *International Journal of Systems Science*, Vol. 29, No. 10, pp. 1065–1080, 1998.

[34] L. E. Parker. ALLIANCE: An Architecture for Fault Tolerant Multi-Robot Cooperation. *IEEE Transactions on Robotics and Automation*, 14(2), pp. 220–240, 1998.

[35] P. J. G. Ramadge and W. M. Wonham. The Control of Discrete Event Systems. it Proceedings of the IEEE, Vol. 77, No. 1, pp. 81–97, January 1989.

[36] M. Rokonuzzaman, R. D. Hale, F. Hwang, J. King, and R. G. Gosine. Modeling, Controlling, and Monitoring Telerobotics Based Mine Operations as Discrete Event Systems. *Proceedings of the 31st International Symposium on Robotics*, Montreal, Quebec, May 2000.

[37] K. Singh and K. Fujimura. A Navigation Strategy for Cooperative Multiple Mobile Robots. *Proceedings of the 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Yokohama, Japan, pp. 283–288, July 1993.

[38] A. Stentz, J. Bares, S. Singh, and P. Rowe. A Robotic Excavator for Autonomous Truck Loading. *Autonomous Robots*, 7, 2, pp. 175–186, 1999.

[39] A. Taholakian and W. M. M. Hales. PN⇔PLC: A Methodology for Designing, Simulating, and Coding PLC Based Control Systems Using Petri Nets. *International Journal of Production Research*, Vol. 35, No. 6, pp. 1743–1762, 1997.

[40] M. Zhou and E. Twiss. Design of Industrial Automated Systems Via Relay Ladder Logic Programming and Petri Nets. *IEEE Transactions on Systems, Man, and Cybernetics — Part C: Applications and Reviews*, Vol. 28, No. 1, pp. 137–150, February 1998.

[41] M. Zhou, Ed. *Petri Nets in Flexible and Agile Automation.* Massachusetts: Kluwer Academic Publishers, 1995.

[42] W. M. Zuberek. Timed Petri nets—definitions, properties, and applications. *Microelectronics and Reliability* (Special Issue on Petri Nets and Related Graph Models), Vol. 31, No. 4, pp. 627–644, 1991.

[43] W. M. Zuberek. *Timed Petri Nets—An Introduction.* Lecture Notes for CS–6726, Memorial University of Newfoundland, 1998.