The Mapping of a Colored Petri Net Model to an OPS5 Rule-Based Production System

Sally S. Lockwitz
Western Michigan University

Follow this and additional works at: http://scholarworks.wmich.edu/masters_theses
Part of the Computer Sciences Commons

Recommended Citation
http://scholarworks.wmich.edu/masters_theses/1031

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master's Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
THE MAPPING OF A COLORED PETRI NET MODEL
TO AN OPS5 RULE-BASED PRODUCTION SYSTEM

by

Sally S. Lockwitz

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
April 1990
THE MAPPING OF A COLORED PETRI NET MODEL TO AN OPS5 RULE-BASED PRODUCTION SYSTEM

Sally S. Lockwitz, M.S.
Western Michigan University, 1990

This paper defines a method by which the colored petri net model of a concurrent system may be mapped to an equivalent rule-based production system. The mapping method defined is an intuitive process which results in a production system that clearly represents the petri net model.

In order to demonstrate the effectiveness and correctness of this method, the colored petri net representation of a part of a flexible manufacturing system of the Renault Car Company is mapped to an equivalent Sierra OPS5 rule-based production system.

An overview of petri nets and rule-based production systems is presented, along with a brief discussion of flexible manufacturing systems (FMSs) and the advantages of using colored petri nets to model FMSs.
ACKNOWLEDGEMENTS

I wish to thank my thesis advisor, Dr. Ben Pinkowski, and my committee members, Dr. Donna Kaminski and Dr. Robert Trenary, for their guidance and direction on this project.

I would also like to thank Dr. Dionysios Kountanis and Reza Rashidi for their help and guidance.

Finally, I would like to thank my partner on this project, Donald Mitchell, for his dedication and contribution to this effort.

Sally S. Lockwitz
INFORMATION TO USERS

The most advanced technology has been used to photograph and reproduce this manuscript from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI
University Microfilms International
A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor MI 48106-1346 USA
313 761-4700 800 521-0600

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
The mapping of a colored petri net model to an OPS5 rule-based production system

Lockwitz, Sally S., M.S.

Western Michigan University, 1990
# TABLE OF CONTENTS

ACKNOWLEDGEMENTS .............................................. ii

LIST OF FIGURES ............................................... vi

CHAPTER

I. INTRODUCTION ......................................... 1

II. REVIEW OF RELATED LITERATURE ......................... 5

III. PETRI NETS ........................................... 12
   Basic Concepts ................................... 12
   Basic Petri Net Example .......................... 13
   Colored Petri Nets ............................... 15
   Colored Petri Net Example ........................ 16

IV. RULE-BASED PRODUCTION SYSTEMS ....................... 18
   Basic Concepts ................................... 18
   Working Memory ................................... 18
   Production Rules ................................ 18
   The Inference Engine ............................. 19
   Reasoning Strategies ............................ 21
   Sierra OPS5 ...................................... 22
   Data Types and Structures ..................... 23
   Production Rules ............................... 24
   Conflict Resolution Strategies ................. 27
Table of Contents - Continued

CHAPTER

V. THE RENAULT CAR COMPANY FMS. ......................... 30
   The Layout and Function of the Flexible Workshop........ 30
   Petri Net Representation of the Workstation Subsystem.... 33
   Colored Petri Net Representation of the Transport Subsystem.... 36
   Colored Petri Net Representation of the Coordination Subsystem........................ 39

VI. FROM A COLORED PETRI NET TO AN OPS5 PRODUCTION SYSTEM......................... 41
   General Mapping Method.................................. 41
   Selection of Data Structure.................................. 41
   Initialization................................................. 42
   Mapping Transitions to Production Rules...................... 43
   The Places in the Fused Colored Petri Net.................. 43
   The OPS5 Representation..................................... 45
   Data Structure................................................. 45
   Initialization................................................. 46
   Production Rules.............................................. 49

VII. EXTENSIONS OF THE PETRI NET........................................ 74
   Modified Petri Net............................................. 74
   Mapping Transitions to Production Rules...................... 74

VIII. CONCLUSION...................................................... 77
Table of Contents – Continued

APPENDIX

A. OPS5 Program Source Code

BIBLIOGRAPHY

81

89
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Petri Net With T1 Enabled</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>Petri Net After Firing of T1</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>Petri Net After Firing of T2</td>
<td>15</td>
</tr>
<tr>
<td>4</td>
<td>Colored Petri Net Example</td>
<td>17</td>
</tr>
<tr>
<td>5</td>
<td>Layout of the Flexible Workshop</td>
<td>30</td>
</tr>
<tr>
<td>6</td>
<td>Path Followed by a Car Body</td>
<td>31</td>
</tr>
<tr>
<td>7</td>
<td>Colored Petri Net Model of the Station Subsystem</td>
<td>34</td>
</tr>
<tr>
<td>8</td>
<td>Colored Petri Net Model of the Transport Subsystem</td>
<td>36</td>
</tr>
<tr>
<td>9</td>
<td>Colored Petri Net Obtained by Fusing CPN's of Figures 7 and 8</td>
<td>40</td>
</tr>
<tr>
<td>10</td>
<td>Extension of Colored Petri Net to Allow Car Body to be Moved From the Workplace Directly to the Roller Table</td>
<td>75</td>
</tr>
</tbody>
</table>
CHAPTER I
INTRODUCTION

Today's complex world of manufacturing poses many challenges for the manufacturing and software engineering disciplines. Among those challenges is the development of control systems for Flexible Manufacturing Systems (FMS's).

A flexible manufacturing system (FMS) is a modern manufacturing method which is an integrated computer controlled cell or area consisting of workstations in which the manufacturing of products takes place, and a transport subsystem which carries the product to and from the workstations. The control system of an FMS consists of the following (Martinez, Muro and Silva 1987):

1. Local controllers for each of the components of the FMS.
2. A coordination system which coordinates the activities of the components of the FMS.
3. A real time scheduling system which can provide rescheduling in case of failures in the FMS or to satisfy other rescheduling requirements.
4. A high level production scheduling system.
5. A production planning system.

The task of developing software systems for the control of flexible manufacturing systems can be simplified by selecting an
appropriate tool for modeling the FMS. The tool selected must be powerful enough to allow for the development of a model which accurately describes the complex interactions in the FMS. The resulting model must provide an easy visualization of the FMS and lend itself to systematic validation.

Considerable research has been conducted on the subject of using colored petri nets as models for the construction of concurrent systems, such as FMS's (see chapter II). This research has shown that colored petri nets are an effective and powerful tool for modeling these systems. Colored petri nets are easily understood as they are graphic in nature, and the constructed model may be analyzed mathematically to determine if there are deficiencies or errors in the model.

Once a colored petri net model has been constructed, a method must be defined by which the modeled system can be developed for production use. The focus of this paper is the development of a method for mapping a colored petri net model of an FMS to a rule-based production system. The rule-based production system can then be used as the control system for the FMS, or as a simulator through which the behavior of the FMS can be monitored and analyzed. The rule-based production system may also be used as another model for further study and analysis.

The method developed in this thesis results in generation of production system working memory elements which can be easily derived from the initial markings on the petri net. This method
also permits a direct mapping from the transitions in the colored petri net model to production rules, which clearly perform the functions specified by the transitions in the petri net.

In order to demonstrate the effectiveness and correctness of the mapping method defined, a sample case is presented. The colored petri net model representing the coordination system for a part of a flexible manufacturing workshop of the Renault Car Company is mapped to an equivalent Sierra OPS5 production system using the mapping method defined in this paper (Martinez and Silva 1984, Rashidi and Kountanis).

Though the discussion in this paper deals primarily with the mapping of a colored petri net model for the coordination system of an FMS, the method defined is applicable to a colored petri net model of any part of an FMS or to a petri net model of any other concurrent system.

Chapter II provides a brief overview of related research. An overview of petri nets and rule-based production systems is presented in Chapters III and IV. Chapter V provides background information on the Renault Car Company FMS and the petri nets developed by Martinez and Silva (1984) to represent the FMS. Chapter VI presents the method developed in this thesis for mapping a colored petri net model to a rule-based production system, and demonstrates the correctness of this method by showing how one petri net example may be mapped to an equivalent OPS5 production system. Chapter VII provides an extension to the petri net under
consideration, and demonstrates how this extension is mapped to production rules. Appendix A contains the complete Sierra OPS5 implementation of the petri net model of the coordination system for the Renault FMS.
CHAPTER II

REVIEW OF RELATED LITERATURE

Considerable research has been conducted on the use of petri nets and their application to the modeling of asynchronous concurrent systems such as flexible manufacturing systems. The review of related research presented in this chapter summarizes the more recent literature available as it relates to the subject of this paper.

Alla, Ladet, Martinez, and Silva-Suarez (1985) demonstrate that a First In First Out (FIFO) queue may be modeled with a colored petri net with two places. A colored petri net model of a flexible manufacturing system for the Renault Car Company is constructed, validated, and shown to be dead-lock free and bounded.

Banaszak and Abdul-Hussin (1987) detail the use of petri nets in automatic design of control programs which supervise concurrent, pipeline-like processes in production systems. The results of their efforts show that this approach is viable both for designing a computer aided planning system and for real-time industrial controllers.

Beck and Krogh (1986) present a method for the construction of petri net models of manufacturing processes used to simulate and design discrete control logic. A model of a two-arm robotic
assembly cell is constructed as an example.

Bourey et al. (1986) present a methodology for the analysis and synthesis of the control of flexible manufacturing cells using structured, adaptive colored petri nets. The primitive functions of a structured adaptive colored petri net are defined and an example of parallel control of a flexible manufacturing cell is presented.

Bruno, Elia, and Laface (1985) present an OPS5 rule-based production system implementation of a decision support system for production planning and real time scheduling of a flexible manufacturing system. The rule based decision support system described is based on a hybrid model which combines and exploits two different methodologies: artificial intelligence and queuing network analysis. The former is used for knowledge representation and problem solving, and the latter for fast performance evaluation capabilities.

Castelain, Corbeel, and Gentina (1985) compare two methods of dynamic simulation of interpreted adaptive petri nets. The first approach presented was programmed in the Pascal language and the second used an interpreter written in the Logo language.

Coolahan and Roussopoulos (1983) outline a methodology for specifying timing requirements for a class of embedded systems called time-driven systems. Time-driven systems have a master timing mechanism which controls the performance of activities performed at regular intervals. A petri net with timing is used
for modeling purposes.

Courvoisier et al. (1987) present a methodology for development of a distributed real-time emulation of a flexible manufacturing system using petri nets on a LAN with remote loading into the machine controller. An overview of the "Specification Emulation for Computer Integrated Automation" (S.E.CO.I.A) project is also presented, which is a methodology for the specification and implementation of distributed discrete control systems.

Desclaux et al. (1986) introduce a flexible specification tool called MLC (Meta Language for Control) used to design the control systems for flexible manufacturing cells. MLC is based on the use of petri nets and is a component of the S.E.CO.I.A. project. The advantage of MLC is that different types of petri nets can be used, as appropriate, to represent the various levels of control of the FMS.

Esteban, Valette, and Courvoisier (1986) present three methods for petri net analysis: (1) reduction of the model, (2) reachability trees, and (3) net decomposition into elementary sub-nets. They show that it is possible to use micro-computers to validate the logical controller behavior in an FMS modeled with a petri net.

Fox and Smith (1984) describe ISIS, which is an artificial intelligence implementation of an interactive job shop scheduling system capable of incorporating all relevant constraints of the system in order to produce realistic job shop production schedules.
They investigate both the representation of the constraints and the ways in which the constraints interact during the search for an acceptable schedule.

Gentina and Corbeel (1987) propose a method for the automatic design of hierarchical control systems for flexible manufacturing systems. Their approach uses structured, colored adaptive petri nets to model low level control, and an expert system to design the petri nets and describe the rules for the control strategy.

Harhen, Ketcham, and Browne (1986) review recent developments in simulation theory, such as an improved user interface, use of a structured approach to developing simulation code, use of hierarchical models to decompose a system into sub-models, and use of file management tools. They then explore the uses of artificial intelligence and its application to the process of constructing simulation models of manufacturing systems. An experiment using frame based software to develop a simulation model is described.

Kamath and Viswanadham (1986) discuss the use of petri nets in the design and analysis of control systems for flexible manufacturing systems. This discussion includes the use of petri nets for FMS simulation, deadlock detection, and FMS performance evaluation. They conclude that the compactness of colored petri nets and their capability for fast and efficient computation of invariants used to detect the presence of deadlocks makes them an ideal tool for FMS modeling.

Leiden et al. (1987) developed a prototype expert system
architecture in which knowledge is represented in goal/plan procedural nets and implemented using petri nets. The system developed actually constructs the procedural nets from a group of rules and plans in its knowledge base, and then executes the nets and resolves any conflicts.

Martinez, Alla, and Silva (1986) describe the use of petri nets and colored petri nets for the modeling of a flowshop production system and a transport system which uses automatic guided vehicles. The advantages and disadvantages of the two petri net types are discussed, as well as the use of the petri net models for simulating various layouts for the FMS.

Martinez, Muro, and Silva (1987) discuss the use of colored petri nets as a modeling tool for the design of the coordination subsystem of a flexible manufacturing system. The integration of the coordination subsystem into the control system of the FMS is also presented. Communication between the components of the FMS is achieved through the information contained on the colors defined on the coordination model. They also propose the use of an expert system as the real-time scheduling subsystem.

Martinez and Silva (1984) discuss many of the problems encountered in controlling a flexible manufacturing system, and the use of colored petri nets for modeling such systems. They outline methods for developing and validating a colored petri net model and develop such a model for a portion of a flexible manufacturing workshop of the Renault Car Company. The petri net model developed
by Martinez and Silva is the model which is mapped to a rule-based production system in this thesis. Martinez and Silva then present a language for describing systems modeled by colored petri nets.

Nelson, Haibt, and Sheridan (1983) describe a method for translating annotated petri nets into a language called "XL/1" which can be optimized and translated into existing compiler languages. In this way, the petri nets can be used as machine-processable representations of process control systems, which can be interactively fired to facilitate debugging of the design.

Rashidi and Kountanis (1989) discuss the use of petri nets as a model for the design of flexible manufacturing systems and the possible relationship of this type of model with artificial intelligence representation techniques. The coordination system for a portion of the flexible manufacturing workshop of the Renault Car Company is modeled with several petri nets. The possibilities for simulation of the coordination system of a flexible manufacturing system in a production based language is discussed. The petri net examples presented in the paper by Rashidi and Kountanis are the same as those presented in the paper by Martinez and Silva (1984), and are also the ones presented in this thesis. The paper by Rashidi and Kountanis provided the basis for the development of a method for mapping petri nets to rule-based production systems.

Zisman (1978) proposes a control structure for modeling
individual processes and a petri net to model the relationships between processes.
CHAPTER III

PETRI NETS

Basic Concepts

A petri net is a special case of a bipartite directed graph consisting of three types of components: places, transitions, and arcs. Places represent states or conditions of the system and are represented by circles. Transitions represent events, which change the state of the system, and are represented by bars. Directed arcs which connect the places and transitions in the colored petri net represent the changes which will take place in the system as a result of an event occurring. The occurrence of an event in a petri net is referred to as a transition firing.

Places in the petri net may contain tokens. To indicate the presence of a token on a place, a dot is placed inside the appropriate circle. A petri net with tokens on its places is referred to as a marked petri net. A vector is used to indicate the current marking of a petri net, or the current state of the system. The vector consists of a list of places and tokens on those places. The presence of a token on a place indicates that the condition represented by that place has been satisfied. The absence of a token on a place indicates that the condition is not satisfied.
A directed arc from a place to a transition indicates that a token is removed from the place when the transition fires. The place is referred to as an input place of the transition. A directed arc from a transition to a place indicates that a token is put on that place when the transition fires. In this case, the place is referred to as an output place of the transition. In order for a transition to fire, the transition must be enabled. A transition is enabled when each input place of the transition has at least one token on it. When a transition fires, a token is removed from each of its input places and a token is placed on each of its output places.

Basic Petri Net Example

Figure 1 is an example of a marked petri net containing four places: P1, P2, P3, and P4; and three transitions: T1, T2, and T3. Places P1 and P2 are input places to transition T1. Places P3 and P4 are the output places of transition T1. P3 is an input place to transition T2, and place P1 is an output place of transition T2. Place P4 is an input place of transition T3, and place P2 is an output place of transition T3.

Initially places P1 and P2 each contain one token. T1 is enabled and may fire, as both of its input places contain tokens. After the firing of T1, the petri net looks like that shown in Figure 2. Places P3 and P4 now each contain one token and the tokens that were on P1 and P2 have been removed. Transitions T2
and T3 are now enabled as shown in Figure 2.

Figure 1. Petri Net With T1 Enabled.

Figure 2. Petri Net After Firing of T1.

If transition T2 fires, the token on P3 is removed and placed on P1, as shown in Figure 3. In Figure 3, T3 is the only
transition that is enabled. T1 is not enabled as P2 is also an input place for T1, and P2 does not contain a token.

If transition T3 fires, the token on P4 is removed and placed on P2. Once transitions T2 and T3 have both fired, the petri net will again look like that shown in Figure 1, with only transition T1 enabled.

![Petri Net After Firing of T2.](image)

Colored Petri Nets

A generalized petri net allows multiple arcs to and from a place in the net. This implies that more than one token may be placed on or removed from a place when a transition fires.

The type of model used in this paper is a generalized colored petri net, which in addition to allowing multiple arcs, associates a color with each token. A set of colors is associated with each
place, indicating the token colors allowed on that place. At any point in time, the color set associated with a place will indicate the colors present on that place. The arcs from a place to a transition are labeled with a function, set, or color indicating which tokens are to be removed from the input place if the transition fires. Arcs from transitions to places are labeled with functions, sets, or colors indicating the tokens to be placed on its output places when the transition fires.

**Colored Petri Net Example**

Figure 4 depicts a colored petri net with three places: P1, P2, and P3; and two transitions: T1 and T2. The color set on place P1 is the set $t=(1,2,3,4)$. The color set on place P2 is $s=(1,2,3,4)$. Place P3 contains markings of the form $<s_i^t_j>$. The output arc from P1 to T1 indicates that when T1 fires, a token of color $t_j$ is removed from P1. The output arc from P2 to T1 indicates that when T1 fires, a token of color $s_i$ is removed from P2. The output arc from transition T1 to place P3 indicates that the color set $<s_i^t_j>$ will be placed on place P3. When transition T2 fires, the color set $<s_i^t_j>$ is removed from place P3. The token $t_j$ is returned to place P1 and the token $s_i$ is returned to P2.

Note that if T1 fires 4 times for $i,j$ equals 1 to 4, transition T1 cannot fire again until T2 has fired at least once. Also note that if T1 fires once with $i=x$ and $j=y$, T1 cannot fire.
again with \( i=x \) and/or \( j=y \) until those colors have been returned to P1 and P2 through the firing of T2 with \( i=x \) and \( j=y \).

![Figure 4. Colored Petri Net Example.](image)

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
CHAPTER IV

RULE-BASED PRODUCTION SYSTEMS

Basic Concepts

Throughout this paper, the use of the term production system refers to a program written in a production system language. Production system languages differ from conventional programming languages mainly in that they consist of a collection of data-sensitive, unordered, independent rules, rather than sequenced instructions. A production system generally consists of (a) data, (b) a set of rules, and (c) an inference engine (Wolfgram, Dear, and Galbraith, 1987).

Working Memory

In a production system, the data is usually referred to as working memory. The data contained in working memory represent a collection of facts. Each item in working memory is referred to as an element or as a working memory element.

Production Rules

The set of rules, also referred to as production rules or productions, constitutes the program. Each of the rules in a
production system consists of a condition and an action to be taken if the condition is satisfied. The rules are stored in what is known as production memory (Brownston et al. 1986).

The condition part of the rule, referred to as the left-hand side (LHS) of the rule, or the antecedent, is used to evaluate the facts stored in working memory. The boolean operators "and" and "or" are usually allowed in the left-hand side of a rule. In fact, the left-hand side of a rule is usually a boolean combination of clauses. Each clause places a restriction on the value of some element of working memory.

The action part of the rule, referred to as the right-hand side (RHS) of the rule, or the consequent, consists of a list of actions to be taken if the left-hand side of the rule is satisfied and the rule fires. These actions may be modifications of existing elements of working memory, creation of new elements of working memory, removal of existing elements of working memory, or a combination of the three. In addition to affecting the contents of working memory, the actions specified by the right-hand side of a rule may result in data or messages being read from or written to input/output devices.

The Inference Engine

The third component of a production system is the inference engine, which actually executes or fires the rules. The inference engine contains a rule interpreter that understands the syntax of
the production system language. The inference engine first determines which rules can fire given the current working memory configuration. A rule can fire only if all of the conditions on the left-hand side of the rule are satisfied. Quite frequently, more than one rule can fire for a given working memory configuration. All rules which can fire are collectively known as the conflict set. The process of determining which rule will fire is known as conflict resolution (Wolfgram, Dear, and Galbraith 1987). Once a rule has been selected and executed, the system cycles back to again determine all rules which can fire. This is referred to as the recognize-act cycle. Since the contents of working memory generally change each time a rule fires, the conflict set is usually different each time the machine cycles. The recognize-act cycle will terminate if the conflict set is empty or an explicit halt was executed. Breakpoints can also be set, which will cause the inference engine to stop at the specified breakpoint, or a cycle count can be set to limit the number of cycles the inference engine makes (Brownston et al. 1986).

Elements of working memory may have several attributes associated with them, which are used by the inference engine in determining which rules are most relevant. The first of these attributes is recency. The recency attribute simply indicates when the element was created in working memory or when it was last modified. The assumption is that elements added or modified recently are most relevant. A second attribute is confidence. The
confidence attribute associates a level of certainty about the fact. A third attribute, activation, associates a level of attention that should be devoted to a fact during conflict resolution.

Reasoning Strategies

Production system rules can be applied in either direction depending on the reasoning strategy used by the inference engine. In a forward-chaining system the left-hand side of a rule is matched against working memory but cannot have any impact on the contents of working memory. Working memory contains what is currently known about a problem or situation or the current state of a system. The inference engine matches the left-hand side of the rules against working memory and executes the right-hand side of the rule to update working memory.

Backward-chaining systems work in the reverse direction of forward-chaining systems. In a backward-chaining system, the right-hand side of the rule is used for matching. There is a goal which the system is trying to achieve, and rules are sought which will result in the achievement of that goal. The right-hand sides of the rules represent unattained goals and the left-hand sides represent sub-goals. Goals are broken down into sub-goals until a collection of goals is formed, each of which is attainable immediately or at least represented as simply as possible. The inference engine matches the right-hand side of the rules against
the unattained goals until all goals are attainable or there are no more rules that can be simplified. The solutions to the attainable goals are then combined into an overall solution. Many production systems apply a combination of forward and backward chaining. The production system language used in this paper is a forward chaining system. Therefore, all further discussions in this chapter apply to forward chaining systems.

As can be seen from the description of the inference engine, control in a rule-based system is not based on a transfer of control between rules, nor are rules executed sequentially. The control mechanism is data driven and based on constant re-evaluation of working memory along with the problem solving strategy used by the inference engine.

Sierra OPS5

The production system language selected for this project is Sierra OPS5, which uses a forward chaining strategy.

A Sierra OPS5 program consists of a declaration section that describes the elements of working memory and a production section that contains the production rules. The data are stored in working memory, and the production rules are stored in production memory. The OPS5 inference engine matches condition elements in the left-hand side of the rules against working memory, selects the rule to fire, and then executes the rule by carrying out the actions of the right-hand side of the rule (Brownston et al. 1986).
Data Types and Structures

Two data types, numbers and symbolic atoms, are available in OPS5. Numbers can be signed or unsigned integers or floating point numbers. A symbolic atom is, with few exceptions, any other collection of characters. Sequences that include non-printing characters, spaces, and other symbols such as (,),{,},~, must be bracketed by vertical bars.

A compound data structure, called an element class, is also available in Sierra OPS5. An element class is similar to a record declaration in Pascal. Each component of an element class is called an attribute. An element class declaration has the following format:

\[(4.1) \text{ (literalize } C \text{ attribute 1 attribute 2 attribute 3 \ldots \text{ attribute n}),}\]

where literalize is a Sierra OPS5 command, C is the name of the class, and attribute 1, ..., attribute n are the names of the individual attributes. Comments can be added to the declaration by preceding them with a semicolon. The element class name and each attribute of an element class must be a symbolic atom. In Sierra OPS5, each element of working memory is called an attribute-value
element. It is not necessary to assign a value in the attribute-value element to each of the attributes declared for an element class (Brownston et al. 1986).

Production Rules

Production rules in Sierra OPS5 consist of a unique name, a left-hand side consisting of one or more condition elements, and a right-hand side that consists of a sequence of actions to be taken if all condition elements are satisfied. Each condition element specifies a condition which must be matched against working memory. If all condition elements in a production rule are matched, the rule is said to be instantiated.

When the OPS5 inference engine performs matching on rules, the left-hand side of the rule may allow some attributes of an element of working memory to be ignored if they are not specifically called out in a condition element. In other words, some attributes of an element of working memory may be irrelevant to the production rule under consideration. If this is the case, one may simply leave those attributes out of the condition element and they will be ignored during the match phase.

In addition, attribute variables can be used to cause a condition to be satisfied regardless of the value of the particular attribute. An attribute variable is a symbolic atom surrounded by angle brackets. The value obtained for that attribute, from the working memory element matched, is then bound to that attribute.
variable for the remainder of that production rule. All references to the attribute variable in both the left-hand side and the right-hand side of the rule are bound by the value obtained in the first reference to the attribute variable. In other words, if another condition element uses that attribute variable for an attribute value, an element of working memory will satisfy the condition element only if it has the attribute value bound to that variable. In this way, matches can be restricted by specifying that certain values must be identical in all or some of the condition elements. An attribute variable may also be used in the right-hand side of the rule provided the first reference to the variable was in the left-hand side of the rule. The use of attribute variables provides a means of communication between the left-hand side and right-hand side of a rule, because values obtained from working memory in the left-hand side can be used by actions in the right-hand side.

Condition elements can also consist of operators such as "<", "<=", ">=", ">", "<>", "=" and "<=>" (of the same type). Conjunction and disjunction of condition elements and negation of condition elements other than the first condition element may also be used (Brownston et al. 1986).

Another type of variable, called an element variable, is available. Element variables apply to an entire condition element rather than an attribute. As with an attribute variable, an element variable is a symbolic atom surrounded by angle brackets.
An element variable is bound to an entire element of working memory when the condition element associated with the element variable matches the element of working memory. A specific element variable may appear only once in the left hand side of a rule. The purpose of an element variable is to communicate with the right-hand side of a rule by allowing the right-hand side to refer to a specific element of working memory in order to modify or remove the working memory element. The scope of the element variable is a single rule. To associate a condition element with an element variable, the condition element and the element variable are placed inside curly brackets with the element variable following the condition element.

The right-hand side of a Sierra OPS5 rule consists of a sequence of actions to be taken if the rule was instantiated, i.e., all conditions in the LHS of the rule were satisfied. Each action consists of the name of the action to be taken followed by the arguments to the action. The only action types used in this paper and discussed here are make, modify, and remove.

The make action causes a new element of working memory to be created. The make action has the following format:

\[
\text{(4.2)} \quad \text{(make } C \text{'attribute value ['}\text{attribute value ...}]),
\]

where \( C \) is the name of the element class, attribute is the name of the attribute and value is either a constant or an attribute.
variable. The make statement may also be used at the top level of OPS5 to initialize working memory.

The remove action causes the specified working memory element to be removed from working memory. The remove action arguments are one or more element variables bound to condition elements in the left hand side of the rule. The format for a remove action is

\[(4.3) \ (\text{remove } \langle \text{element variable} \rangle \ \ldots)\].

The modify action is used to modify one or more attributes of an element of working memory. The first argument to the modify action is an element variable indicating which element of working memory is to be modified. The remaining arguments consist of attribute names followed by the new values for those attributes. The value for an attribute can be a constant or an attribute variable bound in the left-hand side of the rule. The format for the modify action is

\[(4.4) \ (\text{modify } \langle \text{element variable} \rangle \ '\text{attribute value} [\ '\text{attribute value} \ \ldots])\]

Conflict Resolution Strategies

In Sierra OPS5 an instantiation consists of the name of the rule that was matched and a list of working memory elements that the rule matched on. Since generally more than one rule is
instantiated, a decision must be made as to which rule will fire. This is referred to as conflict resolution. Two conflict resolution strategies are available in Sierra OPS5: LEX and MEA (Brownston et al. 1986).

The first step in the LEX strategy is a process called refraction. This process eliminates all instantiations from the conflict set which were previously selected and fired. If a working memory element of an instantiation has been modified, it will have a new recency number and will not be deleted from the conflict set as it constitutes a new instantiation.

The second step in the LEX strategy is to order the remaining instantiations by recency. The most recent instantiations are grouped together and the others are removed from the conflict set. If only one instantiation remains, that rule will fire. If more than one instantiation remains and each has only one condition element, the algorithm continues with step 3. Otherwise those instantiations with only one condition element are discarded and the process is repeated with the second largest recency value, the third largest, etc. until the set has only one instantiation or all instantiations have the same number of condition elements. At this point if no single instantiation dominates, the third step is performed.

In the third step, the inference engine orders the remaining instantiations based on the number of tests in all of the conditions of the rule. The idea here is that the rule with the
most tests is more specific and will be selected. If at this point one instantiation does not dominate, the inference engine makes an arbitrary choice between the remaining instantiations.

The MEA strategy works much the same as the LEX strategy, except that it places more emphasis on recency. After instantiations have been removed from the conflict set through the process of refraction, an additional test is performed which compares instantiations based on the recency of the first condition elements. If no single instantiation dominates, the same tests are then performed as in the LEX strategy.

In the Sierra OPS5 production system developed for this paper, working memory contains the current state of all of the components of the system being modeled. The rules, which represent the transitions in the petri net model, act on working memory to continually evaluate and update the state of the system. The LEX conflict resolution strategy was used in this implementation.
CHAPTER V

THE RENAULT CAR COMPANY FMS

The Layout and Function of the Flexible Workshop

Martinez and Silva (1984) have developed a colored petri net model of the coordination system for a portion of the flexible manufacturing workshop of the Renault Car Company. The purpose of the portion of the FMS being modeled is to mastic seal car bodies. The workshop consists of a set of six workstations and a transport system. The layout of the flexible workshop is shown in Figure 5.

![Figure 5. Layout of the Flexible Workshop (Martinez and Silva, 1984).](image)

Each workstation has space for one car body to be worked on and a roller table transfer bench. The transfer bench has two
positions, one for loading a car into the workplace and the other for unloading a car from the workplace (Figure 6).

![Diagram of transport system]

Figure 6. Path Followed by a Car Body (Martinez and Silva, 1984).

The transport system is made up of six roller tables, as shown in Figure 5 and Figure 6. The roller tables serve to transport the cars into the system, to the necessary workstation, and then to transport the car bodies back out of the system after all work has been completed on the car.

For the sake of simplicity, the system being modeled assumes that each workstation is identical and that the time required to perform the operation in each workstation is identical. It is also assumed that the same amount of time is required in each workstation to move a car from the load table to the workplace and from the workplace to the unload table, and that the time required to place a car onto a load table from the transport system and to move a car from the unload table onto the transport system is the same for each workstation. Finally, it is assumed that the same time is required to move a car from one roller table to the next in
the transport system.

In this portion of the flexible workshop, there can be no more than two cars in the system bound for the same workstation. If there are no other cars in the system bound for or residing in workstation x, and a car enters the system bound for station x, it will enter the system on roller table RT1 and progress down the roller tables until it reaches roller table x. It will then be transferred directly into workplace x without stopping on the load table (note it does have to pass over the load table).

If a car enters the system bound for workstation x and there is already a car in the system bound for or residing in workstation x, the car still enters the system on roller table RT1 and progresses down the roller tables until it reaches roller table x. The car is then loaded onto the load table in workstation x where it must wait until work is completed on the car that entered the system first. Once the work is completed on the car in the workplace, that car is moved to the unload table and the car on the load table is loaded into the workplace.

As stated above, when work has completed on a car it is moved from the workplace to the unload table. If the corresponding roller table is empty, the car is then moved onto the roller table and progresses down the transport system until it exits the system from roller table 6. Otherwise, the car must wait on the unload table until the corresponding roller table becomes available.

Since the system being modeled is the coordination system for
the FMS only, no detail is presented on the local control in the workstations. Only the transportation of cars to and from the workstations and the loading and unloading of cars into the workplace are modeled.

Petri Net Representation of the Workstation Subsystem

The first Petri net presented by Martinez and Silva (1984) models the workstations and is shown in Figure 7. Since all six workstations perform the same function, only one Petri net is required to model them. The Petri net in Figure 7 consists of five places (P6, P7, P8, P9, and P10) and seven transitions (t5, t6, t7, t8, t9, t10, and t11).

Place P6 represents the state where there are no cars in the workstation. Initially place P6 contains a colored token for each workstation in the workshop, as the workstations are all empty initially. P7 represents the state where there is one car in the workstation, in the workplace. Place P8 represents the state where there is a car on the unload table of the workstation and no other cars in the workstation. Place P9 indicates the state where there is a car in the workplace of the workstation and a car on the load table. Place P10 indicates the state where there is a car in the workplace and a car on the unload table. These are the only possible states of the workstation.
Transition $t_5$ indicates the event where a car is being loaded into an empty workstation. The firing of this transition causes the workstation to go from state $P_6$ (empty workstation) to state $P_7$ (car in the workplace). Transition $t_6$ represents the event where a car is being loaded onto the load table of a station that already has a car in the workplace. The firing of $t_6$ causes the workstation to go from $P_7$ (car in the workplace) to state $P_9$ (car in the workplace and car on the load table). Transition $t_7$ models
the event where the only car in the workstation is being removed from the workplace and placed on the unload table. Transition t7 causes the workplace to go from state P7 (car in the workplace) to state P8 (car on the unload table). Transition t8 represents the event where a car is being loaded into the workstation while a car resides on the unload table of that station. This transition causes the system to move from state P8 (car on the unload table) to state P10 (car in the workplace and car on the unload table). Transition t9 models the event where a car that was in the workplace is being moved to the unload table and a car that was on the load table is being moved into the workplace. The firing of t9 causes the system to move from state P9 (car in the workplace and car on the load table) to state P10 (car in the workplace and car on the unload table). Transition t10 represents the event where a car is being moved from the unload table onto the transport system roller table, while a car remains in the workstation. The firing of t10 causes the workstation to move from state P10 (car in the workplace and car on the unload table) to state P7 (car in the workplace). Finally, transition t11 represents the event where a car residing on the unload table of a workstation is being moved back out to the roller table in the transport subsystem. Firing of t11 causes the workplace to go from state P8 (car on the unload table) to state P6 (empty workstation).

The labels of the form <s>, next to the places, represent the color sets of the places. In this petri net all places have the
color set \(<s>\), which means station, as the only colors put on these places are the colors (numbers) of the workstations in the state represented by the place. The labels on the arcs are of the form \(<s_i>\) and represent the color (number) of the workstation being removed from a place (state) or put on a place (into a state).

Colored Petri Net Representation of the Transport Subsystem

The second petri net presented by Martinez and Silva (1984) (Figure 8) models the transport subsystem. This petri net consists of five places (P1, P2, P3, P4, and P5) and six transitions (t1, t2, t3, t4, t1, and tU).

![Colored Petri Net Model of the Transport Subsystem](image)

Figure 8. Colored Petri Net Model of the Transport Subsystem (Martinez and Silva, 1984).

Place P1 contains colored tokens indicating the number of cars that may enter the system bound for a workstation at a given point in time. Initially, place P1 contains two colored tokens for each
workstation, since two cars may be in the system for a specific workstation at any given time. Place P2 contains colored tokens representing all empty roller tables in the transport subsystem. P2 is initialized with one colored token for each roller table in the system. Place P3 indicates the state where a car is in the system, on a roller table, bound for a workstation. The color set on place P3 is of the form \( <s_i, t_j> \), where \( s_i \) is the station the car is bound for and \( t_j \) is the roller table on which the car resides. Place P4 represents the state where a car has left the transport system and is in a workplace. The color set on P4 is of the form \( <s_i> \), as the car no longer resides on a roller table. Place P5 represents the state where a car has exited from a workstation, onto a roller table in the transport system, and is ready to be moved out of the system. The color set on P5 is of the form \( <s_i, t_j> \), where \( s_i \) is the station the car was in and \( t_j \) is the roller table on which the car currently resides.

Transition \( t_1 \) represents the event where a car is entering the system and is being placed on roller table \( t_1 \). The firing of \( t_1 \) causes the colored token \( s_i \) to be removed from place P1, indicating that one less car may enter the system bound for workstation \( s_i \). In addition, \( t_1 \) is removed from place P2, indicating that roller table \( t_1 \) is now occupied. The color set \( <s_i, t_j> \) is then put on place P3 to indicate that a car bound for station \( s_i \) is in the system on roller table \( t_1 \).

Transition \( t_2 \) represents the event where a car had work
performed in workstation $s_i$ and is exiting the system from roller table $t_6$. The color set $<s_i, t_6>$ is removed from place P5, indicating that the car is no longer waiting to exit the system. Color $s_i$ is returned to place P1 to indicate that one more car may enter the system bound for station $s_i$. The colored token $t_6$ is returned to place P2 to indicate that roller table $t_6$ is now empty.

Transition $t_3$ represents the event where a car is being moved from one roller table to the next until it reaches the roller table corresponding to the workstation for which it is bound. The firing of transition $t_3$ causes the color set $<s_i, t_j>$ to be removed from P3, as the car no longer resides on roller table $t_j$. It also removes the colored token $t_{j+1}$ from place P2, as this is the roller table to which the car is being moved. Roller table $t_{j+1}$ is, therefore, no longer available. The color set $<s_i, t_{j+1}>$ is then placed on P3, indicating that a car bound for station $s_i$ is now on roller table $t_{j+1}$. In addition, the colored token $t_j$ is returned to place P2, as roller table $t_j$ is now empty.

Transition $t_4$ represents the event where a car that has exited the workstation is being moved from one roller table to the next until it reaches roller table $t_6$. The firing of transition $t_4$ causes the color set $<s_i, t_j>$ to be removed from P5, as the car no longer resides on roller table $t_j$. It also removes the colored token $t_{j+1}$ from place P2, as this represents the roller table to which the car is being moved. Roller table $t_{j+1}$ is, therefore, no longer available. The color set $<s_i, t_{j+1}>$ is then placed on P5,
indicating that a car bound for station \(s_j\) is now on roller table \(t_{j+1}\). In addition, the colored token \(t_j\) is returned to place \(P2\), as roller table \(t_j\) is now empty.

Transition \(t_l\) represents the event where a car bound for workstation \(s^\) resides on roller table \(t^\), and is ready to enter the workstation. When transition \(t_l\) fires, the color set \(<s^,t^>\) is removed from place \(P3\), to indicate that a car bound for station \(s^\) no longer resides on roller table \(t^\). The colored token \(s^\) is then put on place \(P4\) to indicate that a car is now in workstation \(s^\), and the colored token \(t^\) is returned to place \(P2\) indicating that roller table \(t^\) is now empty.

Transition \(t_u\) represents the event where a car that was in workstation \(s^\) is exiting the workstation onto roller table \(t^\). The firing of transition \(t_u\) causes the colored token \(s^\) to be removed from place \(P4\) and the colored token \(t^\) to be removed from place \(P2\). The color set \(<s^,t_j>\) is then put on place \(P5\), indicating that the car now resides on roller table \(t_j\) and is ready to be moved down the transport system to exit.

Colored Petri Net Representation of the Coordination Subsystem

The third petri net presented by Martinez and Silva (1984) is shown in Figure 9. This petri net was obtained by fusing the transitions in Figure 7 and Figure 8 which represent a synchronization between the workstations and the transport subsystem. Transition \(t_l\) (Figure 7) was fused with transitions \(t_5\),
t6, and t8 (Figure 8) and transition tu (Figure 7) was fused with transitions t10 and t11 (Figure 8).

Figure 9. Colored Petri Net Obtained by Fusing CPN's of Figures 7 and 8 (Martinez and Silva, 1984).

It is this third petri net (Figure 9) which is of primary interest in this paper, as this is the petri net model which I have chosen to map to a Sierra OPS5 production system. A detailed description of the places and transitions in the petri net of Figure 9 will be presented, but a method for mapping from a petri net model to a rule-based production system will first be introduced.
CHAPTER VI

FROM A COLORED PETRI NET TO AN OPS5 PRODUCTION SYSTEM

General Mapping Method

The method developed in this paper for mapping a petri net to a production system includes three major phases: selection of the data structure, initialization, and mapping the transitions in the net to production rules.

Selection of Data Structure

Phase 1 involves a decision as to what type of data structure should be used to represent the places in the petri net. For a colored petri net, several options exist. The structure selected in this paper requires that a separate working memory element be created for each color or color set on a place. This provides flexibility in the system developed because the process of adding workstations to the system is relatively simple. One only needs to add five make statements to initialize working memory elements for the new workstation, and modify one rule. Additionally, no complex functions are required to determine which attribute of a working memory element must be referenced or modified. The structure selected also permits a direct translation of the transitions in the petri net to equivalent OPS5 production rules, as will be shown.
later. The markings specified in the petri net in Figure 9 are station and table. Three additional attributes have been added to identify the serial number of the cars in the system and the name of the place. These are car, car2, and name respectively.

One alternate implementation which may more closely approximate the petri net in terms of understanding, is to create one working memory element for each place in the petri net. In this case, an array or list of stations would exist and an array or list of tables would exist in each element of working memory. The absence or presence of a token would be indicated with a null value for that attribute if the token is not present or a value of one if the token is present. Though this may result in a production system which appears to relate more closely to the petri net, it is not as flexible. Changes in the number of workstations could result in changes in every rule in the production system. This also results in a very complex translation of the transitions in the petri net to equivalent OPS5 production rules. The resulting rules are also very difficult to understand.

Initialization

Phase 2 involves the creation of the working memory elements required to represent initial markings on each place in the colored petri net. For the petri net in Figure 9, working memory elements are created to represent the initial markings on places P1, P2, and P6. In Sierra OPS5 this is accomplished with a series of make
Mapping Transitions to Production Rules

Phase 3 involves the actual mapping of each transition in the petri net to production rules. One OPS5 production rule is created for each transition in the petri net. For a given transition, each arc indicating that a token must be present on a place for the transition to fire, must be mapped to a condition element in the production rule which checks for the presence of an element of working memory indicating that such a token or color set exists on that place.

Each arc indicating that a colored token or color set is to be put on or removed from a given place as the result of a transition firing, must be mapped to an action in the right hand side of the rule. An output arc of a transition generally maps to a make statement to create a new element of working memory or a modify statement to change the color set on a place. An input arc maps to a remove statement to delete an element of working memory.

The Places in the Fused Colored Petri Net

The fused petri net representing the coordination system for the flexible manufacturing system being described (Figure 9), consists of 9 places, (P1, P2, P3, P5, P6, P7, P8, P9, P10). The place P4, in the transport subsystem petri net (Figure 8), has been eliminated and replaced with the petri net representing the
workstations. In the petri net of Figure 9, place P1 initially contains two markings for each workstation \(2Es_i\), for \(i=1\) to \(n\), \(n=\)number of workstations), indicating that 2 cars may enter the system for each workstation. When a car enters the system destined for a given workstation, a token is removed from place P1 corresponding to the workstation for which this car is bound. If a car is waiting to enter the system bound for a given workstation and there are no tokens of the appropriate color on place P1, the car cannot enter the system until a token of the appropriate color has been returned to Place P1. Colored tokens are returned to place P1 when a car exits the system from roller table 6.

The markings on place P2 represent a list of empty roller tables in the transport system. Initially, place P2 consists of a token corresponding to each roller table \(Et_i\), for \(i=1\) to \(n\), \(n=\)number of roller tables). Since only one car can occupy a roller table at a time, the marking corresponding to roller table \(t_j\) is removed from place P2 when a car is moved onto roller table \(t_j\). When roller table \(t_j\) again becomes empty, the colored token \(t_j\) is returned to P2.

A token of color \((s_i,t_j)\) on P3 indicates that a car is currently on roller table \(t_j\) in the transport system and is bound for station \(s_i\).

A token of color \((s_i,t_j)\) on P5 indicates that a car which had work performed in station \(s_i\) is now on roller table \(t_j\) in the transport subsystem and is being moved from roller table to roller
table until it is removed from the system.

Place P6 contains colored tokens for each empty station. Initially P6 contains \( (\lambda s_i, \text{ for } i = 1 \text{ to } n, \text{ where } n = \text{number of workstations}) \). A token of color \( s_i \) on P6 indicates that there is no car being worked on in station \( s_i \), nor is there a car waiting on that station’s load or unload table. A token of color \( s_i \) on P7 indicates that there is a car in the workplace of station \( s_i \). There is no car on the load table, nor is there a car on the unload table in that station. A token of color \( s_i \) on P8 indicates that there is only a car on the unload table of station \( s_i \). A token of color \( s_i \) on P9 indicates that there is a car in the workplace of station \( s_i \) and there is a car on station \( s_i \)'s load table. A token of color \( s_i \) on P10 indicates that there is a car in the workplace of station \( s_i \), and there is a car on station \( s_i \)'s unload table.

The OPS5 Representation

The remaining sections of this chapter present concepts specific to Sierra OPS5, along with the implementation of the mapping method defined in this thesis as it is applied to the petri net in Figure 9.

Data Structure

In OPS5, a compound data structure is available, called an element class. The element class is similar to a structure or record structure type in more traditional languages. The element
class is defined by a literalize statement. The structure chosen for the elements of working memory representing the places in the colored petri net of Figure 9 is

\[(6.1) \text{ Name } - \text{Name of the place in the petri net. The places are } (P1,P2,P3,P5,P6,P8,P9,P10).\]

\[\text{Station } - \text{The colors of the stations in the system. For the purpose of the example selected there are 6 stations, } (1,2,3,4,5,6).\]

\[\text{Table } - \text{The colors of the roller tables in the transport subsystem. There are 6 roller tables, one corresponding to each station, } (1,2,3,4,5,6).\]

\[\text{Car } - \text{The serial number of a car in the system.}\]

\[\text{Car2 } - \text{The serial number of a 2nd car, if applicable, in the system.}\]

This structure is defined in Sierra OPS5 with the literalize statement

\[(6.2) \text{(literalize place name station table car car2 )}\]

Initialization

Each element of working memory may contain one or more of the attributes shown in the literalize statement. In OPS5, the make statement creates an element of working memory.

The initial marking of place \(P1\) is \(2Ls_1\). To create this
representation in working memory in Sierra OPS5, the following make statements are used:

\[(6.3) \quad (\text{make place } \text{name p1 } \text{station 1})
\]
\[(\text{make place } \text{name p1 } \text{station 1})
\]
\[(\text{make place } \text{name p1 } \text{station 2})
\]
\[(\text{make place } \text{name p1 } \text{station 2})
\]
\[(\text{make place } \text{name p1 } \text{station 3})
\]
\[(\text{make place } \text{name p1 } \text{station 3})
\]
\[(\text{make place } \text{name p1 } \text{station 4})
\]
\[(\text{make place } \text{name p1 } \text{station 4})
\]
\[(\text{make place } \text{name p1 } \text{station 5})
\]
\[(\text{make place } \text{name p1 } \text{station 5})
\]
\[(\text{make place } \text{name p1 } \text{station 6})
\]
\[(\text{make place } \text{name p1 } \text{station 6})
\]

The initial marking of place P2 is $E_{t_1}$. To create this representation in working memory in Sierra OPS5, the following make statements are used:

\[(6.4) \quad (\text{make place } \text{name p2 } \text{table 1})
\]
\[(\text{make place } \text{name p2 } \text{table 1})
\]
\[(\text{make place } \text{name p2 } \text{table 2})
\]
\[(\text{make place } \text{name p2 } \text{table 2})
\]
\[(\text{make place } \text{name p2 } \text{table 3})
\]
\[(\text{make place } \text{name p2 } \text{table 3})
\]
\[(\text{make place } \text{name p2 } \text{table 4})
\]
\[(\text{make place } \text{name p2 } \text{table 4})
\]
\[(\text{make place } \text{name p2 } \text{table 5})
\]
\[(\text{make place } \text{name p2 } \text{table 5})
\]
\[(\text{make place } \text{name p2 } \text{table 6})
\]
\[(\text{make place } \text{name p2 } \text{table 6})
\]

The initial marking of place P6 is $E_{s_1}$. To create this representation in working memory in Sierra OPS5, the following make statements are used:

\[(6.5) \quad (\text{make place } \text{name p6 } \text{station 1})
\]
\[(\text{make place } \text{name p6 } \text{station 1})
\]
\[(\text{make place } \text{name p6 } \text{station 2})
\]
\[(\text{make place } \text{name p6 } \text{station 2})
\]
\[(\text{make place } \text{name p6 } \text{station 3})
\]
\[(\text{make place } \text{name p6 } \text{station 3})
\]
\[(\text{make place } \text{name p6 } \text{station 4})
\]
\[(\text{make place } \text{name p6 } \text{station 4})
\]
\[(\text{make place } \text{name p6 } \text{station 5})
\]
\[(\text{make place } \text{name p6 } \text{station 5})
\]
\[(\text{make place } \text{name p6 } \text{station 6})
\]
\[(\text{make place } \text{name p6 } \text{station 6})
\]

In order to make a determination as to whether a car can be
moved from one roller table to the next, it is necessary to first
determine the roller table that the car resides on, and then check
to be sure that the next roller table is available. In order to
accomplish this, one must be able to perform computations in the
left-hand side of the rule as it is necessary to compute one plus
the current roller table the car is residing on. In Sierra OPS5,
calculations cannot be made on the left hand side of a production.
To overcome this problem, 5 additional make statements are used.
These are:

(6.6)  (make addl "a 1 "b 2)
       (make addl "a 2 "b 3)
       (make addl "a 3 "b 4)
       (make addl "a 4 "b 5)
       (make addl "a 5 "b 6)

The literalize statement defining this element class is

(6.7)  (literalize addl
        "a
        "b
    )

These elements of working memory allow you to read in an
element of working memory of type place corresponding to a car
residing on a roller table, assigning that table value to the
variable "tb>. Then, a record of working memory of type addl is
read, where "a is equal to "tb>. The value of "b is then bound to
the variable "tb2>. An attempt is then made to read in an element
of working memory of type place with "name P2 and "table "tb2> to
determine if roller table "tb2> is empty. If all of these
conditions are satisfied, then the car can be moved to the next roller table, <tb2>.

One additional place, S, has been added to the petri net in Figure 9. Place S contains a list of colored tokens representing cars waiting to enter the system. The colors on place S are of the form <s_i,c_k> where s_i is the station the car is bound for and c_k is the serial number of the car. To represent this in working memory, statements of the following type are used for each car waiting to enter the system:

(6.8) (make place 'name s 'station s_i 'car c_k).

Production Rules

The transitions in the petri net of Figure 9 are mapped to OPS5 production rules by mapping each input arc of the transition to an equivalent condition element and each output arc to an equivalent action in the RHS of the rules. Each transition and its corresponding Sierra OPS5 production rule is described below. These rules are described in a logical order as opposed to numerical order.

Transition t1 indicates the event where a car enters the system. In the petri net (Figure 9), transition t1 requires that the token t_i be present on P2 and be removed from place P2 and that token s_i be present and be removed from place P1, and that these tokens be placed on place P3. Additionally, the color set (s_i,c_k) must be removed from place S. The marking (s_i,t_i,c_k) is then
placed on place P3. This is accomplished in Sierra OPS5 with the following production rule:

\[(6.9)\]  
\[
(p \text{ firetl_input_body_car} \\
\{(place \text{'name s 'station <st> 'car <sn>} <s>)\} \\
\{(place \text{'name pl 'station <st>} <p1>)\} \\
\{(place \text{'name p2 'table 1} <p2>)\} \\
\rightarrow
\]
\[(remove <s>)\]  
\[(remove <p1>)\]  
\[(remove <p2>)\]  
\[(make place \text{'name p3 'station <st> 'table 1 'car <sn>} <s>)\].

The first condition in the production rule,

\[(6.10)\]  
\[
\{(place \text{'name s 'station <st> 'car <sn>} <s>)\},
\]
checks working memory to see if there is a car waiting to enter the system. The working memory element looks like that in example 6.8. If so, the station it is bound for is saved in the variable <st> and the serial number of the car is saved in the variable <sn>. This is referred to as a binding. The variable <st> is now bound to the value obtained in this condition for the entire rule. Additionally the entire condition element is bound to the element variable <s>. If this condition had failed, it would indicate that there are no cars waiting to enter the system.

The second condition,

\[(6.11)\]  
\[
\{(place \text{'name pl 'station <st>} <p1>)\},
\]
checks to be sure that there are not already two cars in the system bound for station <st>, as defined in the first condition. If
there are not already two cars in the system for station <st>, then this condition element is bound to the element variable <p1>. The working memory element looks like those in example 6.3. In terms of the petri net, place P1 is checked to see if the colored token s₁ is present on P1.

The third condition,

\[ (6.12) \quad \{(\text{place \'name p2 \'table 1) <p2}\}, \]

checks to be certain that table 1 is available. In other words, there is not already a car sitting on table 1. If there is no car on table 1, the condition element is bound to the element variable <p2>. The working memory element looks like that in example 6.4. In terms of the petri net, place P2 is checked for the presence of the colored token t₁.

If all of the above three conditions are satisfied, then the condition element bound to element variable <s> is removed from working memory, in order to remove the car from the waiting list, by executing the following statement:

\[ (6.13) \quad \text{(remove <s>)} \].

The next action,

\[ (6.14) \quad \text{(remove <p1>)} \],

is executed to remove one of the tokens for the station <st> from place P1. This reduces by one, the number of cars that may enter
the system bound for station \(<st>\). If there are no other cars in
the system bound for station \(<st>\), then there will still be one
working memory element of the type in example 6.3 left for this
station. In terms of the petri net, the colored token \(s_1\) is
removed from place \(P1\).

The next action,

\[
(6.15) \ (\text{remove } <p2>),
\]

is executed to remove table 1 from the list of available tables.
In terms of the petri net, \(t_1\) is removed from \(P2\).

Finally, a new element of working memory is created:

\[
(6.16) \ (\text{make place } ^\ast \text{name } p3 ^\ast \text{station } <st> ^\ast \text{table } 1 ^\ast \text{car } <sn>).
\]

This statement creates a working memory element indicating that a
token has been placed on place \(P3\) with colors \((<st>,t_1,<sn>)\). This
means that there is now a car with serial number \(<sn>\) in the system
bound for station \(<st>\) that is currently residing on table 1. In
terms of the petri net, \(s_1\) and \(t_1\) are placed on \(P3\).

Transition \(t_3\) represents the event where a car is being moved
from one roller table to the next in order to get it to the roller
table corresponding to the station it is bound for. In the petri
net, transition \(t_3\) requires that table \(t_j\) be present and be removed
from place \(P3\) and placed on \(P2\) and that table \(t_{j+1}\) be present and
be removed from place \(P2\) and placed on \(P3\). This is accomplished in
Sierra OPS5 with the following production rule:
The first condition,

\[(6.18)\quad ((\text{place } \text{name } p3 \quad \text{station } \text{st} \quad \text{table } \{<tb> < <\text{st}>\}) \quad <p3>)\]  
checks for an element of working memory indicating that there is a car on a roller table that needs to be moved to the next roller table. The place P3 is checked to see if there is a car bound for station \text{st}, that is sitting on a table \text{tb}, where \text{tb} is less than \text{st}. If \text{tb} were equal to \text{st} then the car is already on the correct roller table and should not be moved further. If such an element of working memory exists, the variable \text{st} will contain the station number the car is bound for, \text{tb} will contain the number of the roller table the car currently resides on, and the element variable \text{p3} will be bound to this condition element. In terms of the petri net, P3 is checked for the presence of \(s_i\) and \(t_j\).

The second condition,

\[(6.19)\quad \text{addl } a \quad \text{tb} \quad b \quad \text{tb2})\],
is necessary because computations cannot be performed on the left hand side of a rule. A condition must exist which checks to see if
the next roller table is available. In order to determine what the next table is, given the current table, some computation is required. To overcome this problem, elements of working memory were created as in example 6.6. This condition finds the working memory element of type add1 corresponding to table <tb>. It stores the value of the next roller table in the variable <tb2> for use in the third condition.

The third condition,

(6.20) \{(place "name p2 "table <tb2>) <p2>\},

uses the value <tb2> obtained in the second condition to see if the next table is available. If so, the condition element is bound to the element variable <p2>. In terms of the petri net, place P2 is checked for the presence of \( t_{j+1} \).

If the above three conditions are satisfied then the following occurs.

The first action,

(6.21) \{(modify <p2> "table <tb>\},

causes the token <tb> to be returned to place P2 and the token <tb2>, which represents table <tb+1> to be removed from place P2. This indicates that now table <tb> is available, but table <tb+1> (<tb2>) is occupied. The modify statement in Sierra OPS5 actually results in a remove statement which removes condition element <p2> and a make statement which makes a new element of working memory.
In terms of the petri net, token $t_{j+1}$ has been removed from P2 and token $t_j$ has been returned to P2.

The second action,

\[(6.22) \ (\text{modify } <p3> \ ^\text{table} \ <tb2>),\]

modifies the working memory element $<p3>$ by changing the current roller table from $<tb>$ to table $<tb2>$, the next table. In terms of the petri net, token $t_j$ has been removed from P3 and token $t_{j+1}$ has been placed on P3.

Transition $t_4$ in the petri net represents the event where a car has already been processed in the appropriate workstation and is now being moved from roller table to roller table in the transport system until it finally reaches roller table 6, the last roller table.

Transition $t_4$ in the petri net requires that table $t_j$ be present on and removed from place P5 and placed on P2 and that table $t_{j+1}$ be present on and removed from place P2 and placed on P5. This is accomplished in Sierra 0PS5 with the following production rule:

\[(6.23) \ (p \ \text{fire}t_4 \ \text{next_table_output}
\begin{align*}
&\{\text{(place ^name p5 ^station st ^table }<tb}\rangle = <st}\}) <p5>\} \\
&(\text{addl a }<tb> \ b <tb2>) \\
&\{\text{(place ^name p2 ^table }<tb2}\rangle <p2>\}
\end{align*}\]

\[\text{--->} \]

\[(\text{modify } <p2> \ ^\text{table }<tb>)
(\text{modify } <p5> \ ^\text{table }<tb2>).\]

The first condition in the production rule,
(6.24)  \{(place \text{'name} p5 \text{'station} <st> \text{'table} \{<tb> \geq <st>\}) \text{p5}\},

checks for an element of working memory indicating that there is a car on a roller table that needs to be moved to the next roller table. The place P5 is checked to see if there is a car that was worked on in station <st>, residing on table <tb>, where <tb> is greater than or equal to <st>. If <tb> were less than <st> then the car has not been worked on yet. If such an element of working memory exists, the variable <tb> will contain the number of the roller table the car currently resides on and the element variable <p5> will be bound to this element of working memory. In terms of the petri net, place P5 is checked for the presence of \(s_i\) and \(t_j\).

The second condition,

(6.25)  \{(addl \text{'a} <tb> \text{'b} <tb2>)\},

finds the working memory element of type \text{addl} corresponding to table <tb>. It stores the value of the next roller table in the variable <tb2> for use in the next condition.

The third condition,

(6.26)  \{(place \text{'name} p2 \text{'table} <tb2>) \text{p2}\},

uses the value <tb2> obtained in the second condition, to see if the next table is available. If so, this condition element is bound to the element variable <p2>. In terms of the petri net, place P2 is checked for the presence of the colored token \(t_{j+1}\).
If the above three conditions are satisfied then the following occurs. The first action,

\[(6.27) \quad \text{(modify <p2> table <tb>)} \]

causes the token <tb> to be returned to place P2 and the token <tb2>, which represents table <tb+1>, to be removed from place P2. This indicates that now table <tb> is available, but table <tb+1> (<tb2>) is occupied. In terms of the petri net, token \( t_{j+1} \) has been removed from P2 and token \( t_j \) has been returned to P2.

The second action,

\[(6.28) \quad \text{(modify <p5> table <tb2>)} \]

modifies the working memory element <p5> by changing the current roller table from <tb> to table <tb2>, the next table. In terms of the petri net, token \( t_j \) has been removed from P5 and token \( t_{j+1} \) has been placed on P5.

Transition \( t_2 \) represents the event where all work has been completed on a car in a given station. The car is now residing on table \( t_6 \) and is exiting the system. This transition calls for station \( s_1 \) to be present on P5 and for the colored token \( s_1 \) to be returned to place P1 and for table \( t_6 \) to be returned to place P2. It also calls for \( s_1 \) and \( t_6 \) to be removed from place P5. This is accomplished in Sierra OPS5 with the following production rule:

\[(6.29) \quad \text{(p firet2_output_body_car}
\begin{verbatim}
  ((place "name p5 "station <st> "table 6 "car <sn>) <p5>)
\end{verbatim}
\]
The single condition element,

\[(\text{place } \text{name p5 station st} \text{ table 6 car sn}) <p5>\],

checks working memory to see if a car is in state P5 in the petri net and resides on roller table 6. If so, the condition element is bound to the element variable \(<p5>\). A car in state P5 residing on table 6 indicates that the car has had all work completed on it and is ready to leave the system. In terms of the petri net, P5 is checked for the presence of a car on table \(t_6\) on which service was performed in station \(s_1\). If this is the case, then the first right hand side action,

\[(\text{remove } <p5>)\],

is executed. The remove statement completely removes this element of working memory. This corresponds to removing \(s_1\) and \(t_6\) from place P5 in the petri net.

The second action,

\[(\text{make place } \text{name p1 station st})\],

returns the token \(<st>\) to place P1. The return of \(<st>\) to place P1 indicates that oneless car is in the system for the station \(<st>\). This corresponds to returning \(s_1\) to place P1 in the petri net.
The final action,

\[(6.33) \quad \text{(make place \ 'name p2 \ 'table 6),}\]

returns table 6 to place P2. This indicates that table 6 is now available (empty). This corresponds to returning \( t_6 \) to P2 in the petri net.

Transition \( t_3-t_1 \) in the petri net represents the event where an empty station is being loaded with a car. There is nothing on the load table or unload table of the station. The petri net indicates that colored token \( s_i \) must be present on place P6 and that the color set \( (s_i,t_i) \) must be present on place P3. It then calls for the removal of \( s_i \) from place P6 and the removal of the color set \( (s_i,t_i) \) from place P3. Next, it calls for the placement of \( s_i \) on P7 and the placement of token \( t_i \) on P2. This is accomplished in Sierra OPS5 with the following production rule:

\[(6.34) \quad (p \ \text{fire}t_3 \ t_1 \ \text{load_empty_station}
\begin{align*}
&\{(\text{place \ 'name p3 \ 'station } \text{<st>} \ 'table } \text{<st>} \ 'car \ \text{<sn}> \ <p3>) \ <p3>)
&\{(\text{place \ 'name p6 \ 'station } \text{<st>}) \ <p6>)
\end{align*}
\longrightarrow
\begin{align*}
&(\text{remove } <\text{p6}>)
&(\text{remove } <\text{p3}>)
&(\text{make place \ 'name p7 \ 'station } \text{<st>} \ 'car \ \text{<sn>})
&(\text{make place \ 'name p2 \ 'table } \text{<st>])).
\end{align*}
\]

The first condition element,

\[(6.35) \quad \{(\text{place \ 'name p3 \ 'station } \text{<st>} \ 'table } \text{<st>} \ 'car \ \text{<sn>}) \ <p3>)\},
\]

checks for the presence of the color set \( (s_i,t_i) \) on place P3. Note
that the number of the station must be equal to the number of the roller table in order for the car to be ready to be moved into the station. If this color set is present on P3, then the station $s_i$ is bound to the variable $st$. Since $s_i$ has already been bound to the variable $st$ and the binding is fixed for the entire rule, the number of the table $t_i$ must be equal to $st$ as well. The condition element is then bound to the element variable $p3$.

The second condition,

(6.36) \{(place name p6 station $st$) $p6$\},

checks for the presence of the colored token $s_i$ on the place P6. If token $s_i$ is on P6, then the workstation is empty. Additionally, the condition element is bound to the element variable $p6$.

If both of these condition elements are satisfied then the following action statements are executed:

(6.37) (remove $p6$),

removes the condition element $p6$, indicating that the station is no longer empty. In terms of the petri net, this serves to remove the colored token $s_i$ from P6.

The second action,

(6.38) (remove $p3$),

causes the condition element $p3$ to be removed from working memory to show that the car is no longer on a roller table. In terms of
The petri net, the color set \((s_i, t_i)\) is being removed from place P3.

The third action,

\[(6.39) \quad (\text{make place "name p7" station <st> "car <sn>}),\]

creates an element of working memory of type place with the name P7 with a colored token <st>. This indicates that there is now a car in the workplace of station <st>. In terms of the petri net, this represents the placement of token \(s_i\) on place P7.

The final action,

\[(6.40) \quad (\text{make place "name p2" table <st>}),\]

results in the return of table <st> to place P2, indicating that table <st> is again available. In terms of the petri net, this is the same as moving token \(t_i\) to place P2.

Transition t6-t1 represents the situation where there is already a car in the workstation being worked on and a car is being loaded from the roller-table in the transport subsystem onto the load table in that workstation. This transition requires that the color set \((s_i, t_i)\) be present on place P3, indicating that there is a car residing on roller table \(t_i\) which is ready to be moved into station \(s_i\) and that the colored token \(s_i\) be present on P7, indicating that there is currently a car being worked on in station \(s_i\). If these conditions are met, the color set \((s_i, t_i)\) is removed from place P3 and the colored token \(s_i\) is removed from P7. The
colored token $s_i$ is then placed on place P9, as this is the state representing a car in the workplace and a car on the load table. Additionally, the colored token $t_j$ is placed on P2 indicating that this roller table is now empty. To accomplish this in Sierra OPS5, the following production rule is used:

\[
(6.41) \quad (p \text{ fire}t_6 -t_l \text{ load working station}
\quad \{(\text{place } \text{ name } p_3 \text{ station } <st> \text{ table } <st> \text{ car } <sn_1>) <p_3>\}
\quad \{(\text{place } \text{ name } p_7 \text{ station } <st> \text{ car } <sn_2>) <p_7>\}
\rightarrow
\quad \text{(remove } <p_7>)
\quad \text{(remove } <p_3>)
\quad \text{(make place } \text{ name } p_2 \text{ table } <st>)
\quad \text{(make place } \text{ name } p_9 \text{ station } <st> \text{ car } <sn_1>
\quad \text{car2 } <sn_2>).
\]

The first condition element,

\[
(6.42) \quad \{(\text{place } \text{ name } p_3 \text{ station } <st> \text{ table } <st> \text{ car } <sn_1>) <p_3>\}.
\]

checks for the presence of the color set $(s_i, t_j)$ on place P3 indicating that there is a car on roller table $t_j$ ready to move into station $s_i$. If so, the station $s_i$ is bound to the variable $<st>$. As was explained earlier, the table number must match the station number as the variable $<st>$ is bound to the station number, and so must be equal to the table $<st>$. In addition, the condition element is bound to the element variable $<p_3>$.

The second condition element,

\[
(6.43) \quad \{(\text{place } \text{ name } p_7 \text{ station } <st> \text{ car } <sn_2>) <p_7>\},
\]
checks for the presence of the token $s_1$ on place $P7$, indicating that there is a car in station $s_1$ that is currently being worked on. If so, the condition element is bound to the element variable $<p7>$.

If both of the above condition elements are satisfied then the following action statements are executed.

(6.44) \((\text{remove } <p7>)\)

causes the colored token $s_1$ bound by the variable $<st>$ to be removed from $P7$, indicating that the station no longer has only one car in the workstation.

The second statement,

(6.45) \((\text{remove } <p3>)\),

causes the color set \((s_1, t_1)\) to be removed from place $P3$, indicating that there is no longer a car on table $t_1$ waiting to move into station $s_1$.

The third statement,

(6.46) \((\text{make place } \text{name } p2 \text{ } \text{table } <st>)\),

causes the colored token $t_1$ to be returned to place $P2$, indicating that table $t_1$ is now empty.

The final statement,

(6.47) \((\text{make place } \text{name } p9 \text{ } \text{station } <st> \text{ } \text{car } <sn1> \text{ } \text{car2 } <sn2>)\),
causes the token $s_i$ to be placed on $P_9$, indicating that there is now a car in the workplace and a car on the load table of station $s_i$.

Transition $t_7$ represents the situation where there was a car being worked on in station $s_i$ and nothing was on the load or unload table. The work has been completed and the car is being moved from the workplace to the unload table in station $s_i$. The petri net indicates that the colored token $s_i$ should be present on and removed from $P_7$ and placed on $P_8$, the state indicating that a car is on the unload table in station $s_i$, with nothing on the load table and nothing in the workplace. To accomplish this in OPS5 the following production rule is used:

\[
(6.48) \quad (p \ firet7 \_endwork \_incomplete \_station
\quad \{(\text{place } \_\_\text{name } p7 \_\_\text{station } <st> \_\_\text{car } <sn>) \_\_\_p7)\}
\quad \rightarrow
\quad (\text{remove } <p7>)
\quad (\text{make place } \_\_\text{name } p8 \_\_\text{station } <st> \_\_\text{car } <sn>)).
\]

The single condition element,

\[
(6.49) \quad \{(\text{place } \_\_\text{name } p7 \_\_\text{station } <st> \_\_\text{car } <sn>) \_\_\_p7),
\]

checks place $P_7$ to see if the colored token $s_i$ is present, indicating that there is a car being worked on in station $s_i$ and nothing on station $s_i$'s load or unload table. If so, the station $s_i$ is bound to the variable $<st>$ and the condition element is bound to the element variable $<p7>$.

The first right hand side statement,
causes the colored token $s_i$ to be removed from place $P7$, indicating that there is no longer a car being worked on in station $s_i$.

The second statement,

(6.51) (make place "name p8 "station <st> "car <sn>),

causes the colored token $s_i$ to be placed on P8, indicating that the car that was in the workplace of station $s_i$ is now on the unload table of station $s_i$.

The transition $t8-tl$ represents the situation where a car is on the unload table of station $s_i$ and there is a car waiting on roller table $t_i$ in the transport system to enter the workplace of station $s_i$. The petri net indicates that the colored token $s_i$ must be on place P8 indicating a car on the unload table of station $s_i$ and that the color set ($s_i,t_i$) must be present on place P3, indicating that a car is waiting on roller table $t_i$ to enter station $s_i$. The petri net requires that the colored token $s_i$ must be removed from place P8 and that the color set ($s_i,t_i$) must be removed from place P3. It then requires that the colored token $s_i$ be placed on P10, indicating that there is now a car in the workplace of station $s_i$ and a car waiting on the unload table of station $s_i$, and the colored token $t_i$ be placed on P2 indicating that table $t_i$ is now empty. To accomplish this the following production rule is used:
The first condition element,

(6.53)  ((place `name p3 `station `<st>` `table `<st>` `car `<sn1>) `<p3>),

checks for the presence of the color set (s^,t^) on place P3, indicating that there is a car on table t^, waiting to enter station s^.

If so, the station s^ and the table t^ which are equal are bound to the variable `<st>` and the condition element is bound to the element variable `<p3>`.

The second condition element,

(6.54)  ((place `name p8 `station `<st>` `car `<sn2>) `<p8>),

checks to see if the colored token s_i is present on P8, indicating that there is a car on the unload table in station s_i.

If so, the condition element is bound to the element variable `<p8>`.

If the above two conditions are satisfied then the following action statements are executed:

(6.55)  (remove `<p8>)

removes the token s_i from place P8, indicating that there is no
longer just one car in the workstation.

The statement,

(6.56) (remove <p3>),

removes the color set \( (s_i, t_i) \) from place P3, indicating that there is no longer a car waiting on roller table \( t_i \), to enter workstation \( s_i \).

The third statement,

(6.57) (make place "name p10 "station <st> "car <sn1> "car2 <sn2> ),

places the colored token \( s_i \) on place P10, indicating that there is now a car in the workplace of station \( s_i \) and a car on station \( s_i \)'s unload table.

The last statement,

(6.58) (make place "name p2 "table <st> ),

causes the table \( t_i \) to be returned to the list of empty roller tables by placing the colored token \( t_i \) on place P2.

Transition \( t_9 \) represents the event where a car was in the workplace of station \( s_i \) and a car was waiting on station \( s_i \)'s load table. The work has been completed on the car in the workplace and the car in the workplace is being moved to the unload table, and the car on the load table is being moved into the workplace. The petri net shows that the colored token \( s_i \) must be present on place P9 and that it must be removed from P9 and placed on place P10. To
accomplish this in Sierra OPS5, the following production rule is used:

(6.59) \((p \text{ firet9 endwork complete station}) \rightarrow (\text{remove } <p9>) (\text{make } \text{place } \text{name } p10 \text{ station } <st> \text{ car } <sn1> \text{ car2 } <sn2>)\).

The condition element,

(6.60) \((\text{place } \text{name } p9 \text{ station } <st> \text{ car } <sn1> \text{ car2 } <sn2>) <p9>\),

checks for the presence of the colored token \(s_1\) on the place \(P9\), indicating that there is a car being worked on in station \(s_1\) and that there is a car waiting on the load table in station \(s_1\). If this condition is met, the colored token \(s_1\) is bound to the variable \(<st>\) and the condition element is bound to the element variable \(<p9>\). The following statements are then executed:

(6.61) \((\text{remove } <p9>)\)

removes the colored token \(s_1\) from place \(P9\), indicating that there is no longer a car being worked on in the workstation, and a car is on the load table of station \(s_1\).

The statement,

(6.62) \((\text{make } \text{place } \text{name } p10 \text{ station } <st> \text{ car } <sn1> \text{ car2 } <sn2>)\),

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
places the colored token $s_i$ on place $P_{10}$ to indicate that now there is a car in the workplace being worked on which was the car that was on the load table and that the car that was in the workplace is now on the unload table.

Transition $t_{10}-tu$ in the petri net represents the situation where there is a car being worked on in station $s_i$ and there is a car on station $s_i$'s unload table. The car on the unload table is now being moved out to the roller table in the transport system, freeing the unload table in station $s_i$. In order for transition $t_{10}-tu$ to fire, the colored token $s_i$ must be present on place $P_{10}$ indicating that there is a car being worked on and a car is on the unload table in station $s_i$. The colored token $t_j$ must be present on $P_2$ indicating that roller table $t_j$ is empty. If these conditions are met, $t_{10}-tu$ fires, removing $s_i$ from $P_{10}$ and $t_j$ from $P_2$. The color set $(s_i,t_j)$ is placed on $P_5$ and the colored token $s_i$ is placed on place $P_7$. This indicates that there is now just a car being worked on in station $s_i$ and that there is a car on roller table $t_j$ which needs to be moved to roller table 6 to exit the system. The following OPS5 production rule accomplishes this:

(6.63) (p fire$t_{10}-tu\_unload\_complete\_station
 ((place \"name p10 \"station <st> \"car <sn1> \"car2 <sn2>\) <p10>)
 ((place \"name p2 \"table <st>) <p2>)

--> (remove <p10>)
(remove <p2>)
(make place \"name p5 \"station <st> \"table <st> \"car <sn2>\)
(make place \"name p7 \"station <st> \"car <sn1>\)).
The first condition element,

\[(6.64) \quad \{(\text{place name plO station st car sn1 car2 sn2}) \quad \langle \text{plO} \rangle,\]

checks to see if the colored token \(s_i\) is present on place PlO indicating that there is a car being worked on in station \(s_i\) and a car on station \(s_i\)'s unload table. If this is the case, then \(s_i\) is bound to the variable \(<st>\) and the condition element is bound to the element variable \(<\text{plO}>\).

The second condition element,

\[(6.65) \quad \{(\text{place name p2 table st}) \quad \langle \text{p2} \rangle,\]

checks to see if roller table \(<st>\) is available in the transport subsystem. If the roller table is available, then the condition element is bound to the element variable \(<\text{p2}>\).

If the previous two conditions are satisfied then the following action statements are executed:

\[(6.66) \quad \text{(remove } <\text{plO}>\)\]

removes the colored token \(s_i\) from PlO, indicating that there is no longer a car on the unload table of station \(s_i\).

The second statement,

\[(6.67) \quad \text{(remove } <\text{p2}>\),\]

removes roller table \(t_i\) from the list of available roller tables, as the car on station \(s_i\)'s unload table is being moved to this
roller table. With respect to the petri net, this removes the colored token \( t_i \) from place P2.

The third statement,

\[(6.68) \quad \text{make place \textit{name p5} \textit{station <st> table <st> car <sn2>,}}\]

places the color set \( (s_i, t_i) \) on place P5 to indicate that there is now a car on roller table \( t_i \) that needs to be moved out of the system.

The last statement,

\[(6.69) \quad \text{make place \textit{name p7} \textit{station <st> car <sn1>,}}\]

places the colored token \( s_i \) on place P7, indicating that now a car is being worked on in station \( s_i \), with nothing on the load or unload table.

The last transition \( t4-tu \) represents the situation where there is a car on the unload table of station \( s_i \) and nothing in the workplace or on the load table. The car is now being moved from the unload table back to the roller table leaving the station empty. The petri net indicates that when \( t4-tu \) fires, \( s_i \) must be present on P8 and \( t_i \) must be present on P2. The colored token \( s_i \) is placed on P6, indicating the workstation is now empty, and the color set \( (s_i, t_i) \) is placed on place P5, indicating that there is a car on roller table \( t_i \) which needs to be moved down the transport subsystem and to exit the system. This is accomplished in Sierra OPS5 as follows:
(6.70)  (p firet4-tu_unload_incomplete_station
  (p8) (p2)
  (p8) (p2)

-->
  (remove <p8>)
  (remove <p2>)
  (make place name p5 'station <st> 'table <st> 'car <sn>)
  (make place name p6 'station <st>)).

The first condition element,

(6.71)  ((place 'name p8 'station <st> 'car <sn>) <p8>),

checks for the presence of the colored token $s_i$ on place P8,
indicating that there is a car on the unload table of station $s_i$,
but nothing in the workplace or on the load table. If this is the
case, the station $s_i$ is bound to the variable <st> and the
condition element is bound to the element variable <p8>.

The second condition,

(6.72)  ((place 'name p2 'table <st>) <p2>),

is then checked to see if roller table $t_i$ is available. In terms
of the petri net, the list of empty tables, P2, is checked for the
presence of the colored token $t_i$. Note that the variable <st> is
used in this condition, as the table which must be available is the
roller table corresponding to the station <st> obtained in
condition element 1. If this condition is met, the condition
element is bound to the element variable <p2>.

If both of these condition elements are satisfied, then the
following statements are executed:

(6.73) (remove <p8>)

causes the colored token \( s_i \) to be removed from place \( P8 \), which indicates that there is no longer a car on the unload table of station \( s_i \).

The second statement,

(6.74) (remove <p2>),

causes the colored token \( t_j \) to be removed from place \( P2 \), indicating that the roller table \( t_j \) is no longer empty.

The third statement,

(6.75) (make place "name p5 "station <st> "table <st> "car <sn>),

places the color set \( (s_i, t_j) \) on place \( P5 \), indicating that there is a car on roller table \( t_j \) in the transport subsystem that is ready to be moved out of the system.

The last statement,

(6.76) (make place "name p6 "station <st>),

returns the colored token \( s_i \) to place \( P6 \), indicating that station \( s_i \) is now completely empty.

Appendix A contains the full Sierra OPS5 implementation of the colored petri net shown in Figure 9.
CHAPTER VII

EXTENSIONS OF THE PETRI NET

Modified Petri Net

The petri net shown in Figure 9 does not permit a car to go directly from the workplace back out onto a roller table when the work has been completed. The reason for this is not known. It may be a function of the workstations, but if this is not the case, the petri net can be modified as shown in Figure 10.

This modification involves the addition of two transitions, t12 and t13, to the petri net, and the addition of two production rules to the production system. Now a car would wait on the unload table only if the workstation's roller table was already occupied.

Mapping Transitions to Production Rules

Transition t12 would result in the following Sierra OPS5 production rule:

\[(7.1) \quad (p \text{ firet12}_\text{unload}_\text{from}_\text{working}_\text{to}_\text{empty} \quad ((\text{place \_name p7 \_station <st> \_car <sn>} <p7>) (\text{place \_name p2 \_table <st>} <p2>)) \rightarrow (\text{remove <p7>}) (\text{remove <p2>}) (\text{make place \_name p6 \_station <st>}) (\text{make place \_name p5 \_station <st> \_table <st> \_car <sn>})).\]
Figure 10. Extension of Colored Petri Net to Allow Car Body to be Moved From the Workplace Directly to the Roller Table.

This results in a check on P7 to see if a car is currently in the workplace of station \( s_i \), which is ready to be unloaded and that the roller table \( t_1 \) is empty. If so, the colored token \( s_i \) is removed from place P7 and the colored token \( t_1 \) is removed from place P2, indicating that the roller table is no longer empty. The colored token \( s_i \) is then placed on place P6, indicating that station \( s_i \) is now empty. In addition, the color set \( (s_i, t_1) \) is placed on place P5 to indicate that the car is now on roller table \( t_1 \) and is ready to be moved down the transport subsystem and out of the system.
Transition \( t_{13} \) would result in the following OPS5 production rule:

\[
(7.2) \quad (p \text{ fire}_{t_{13} \text{ unload from working and load to working}} \hfill \\
\{\{\text{place name } p^9 \text{ station } s^5 \text{ car } s^1 \text{ car}_2 \text{ sn}_2\} \{p^9\}\} \hfill \\
\{(\text{place name } p^2 \text{ table } s^5) \{p^2\}\} \hfill \\
\rightarrow \hfill \\
(\text{remove } \langle p^9 \rangle) \hfill \\
(\text{remove } \langle p^2 \rangle) \hfill \\
(\text{make place name } p^7 \text{ station } s^5 \text{ car } s^1) \hfill \\
(\text{make place name } p^5 \text{ station } s^5 \text{ table } s^5 \text{ car } s^2). \hfill 
\]

This transition allows for the event where a car is in the workplace of station \( s^1 \) and the work has been completed. There is also a car on the load table ready to be loaded into the workplace. The first condition checks to see if there is a car in workstation \( s^1 \) and a car on the load table of station \( s^1 \). The second condition checks to be sure that roller table \( t^1 \) is empty by checking the list of empty tables on \( P^2 \). If these conditions are met, the colored token \( s^1 \) is removed from \( P^9 \) and the colored token \( t^1 \) is removed from \( P^2 \), indicating that the roller table is no longer empty. The colored token \( s^1 \) is then placed on \( P^7 \) to indicate that now there is a car in the workplace of station \( s^1 \) and nothing on the load or unload table. Lastly, the color set \( (s^1, t^1) \) is placed on \( P^5 \) to indicate that there is now a car on roller table \( t^1 \) that needs to be moved down the roller tables and out of the system. Though this modification does not result in any significant changes to the petri net or the OPS5 production system, it does eliminate an operation in the coordination system.
CHAPTER VIII

CONCLUSION

The object of this paper was the development of a method for mapping a colored petri net model to an OPS5 rule-based production system. The most difficult part of this process was the selection of a production system data structure which was easily understood and which clearly represented the petri net, yet allowed for the development of production rules which mapped as directly as possible from the transitions in the petri net.

Many data structures are available which could have been selected to represent the places in the petri net. Each of these data structures results in a different set of production rules to test and manipulate working memory. Numerous representations were investigated, before the one selected was determined to be optimal in terms of the objectives stated above.

There are some minor weaknesses in the representation chosen in terms of flexibility and storage requirements. Adding a workstation to the FMS requires the addition of five working memory elements and the modification of one rule.

Two elements of working memory must be added to reflect that two cars can exist in the system for the workstation being added. A third working memory element must be added to reflect the availability of the additional corresponding roller table and a
fourth working memory element must be added to indicate that the new workstation is initially empty. The fifth element of working memory that must be added is of the type addl, which is used to compute the added roller table as the next table when appropriate.

The addition of a workstation to the FMS also requires that the rule firet2_output_body_car be modified. The number of workstations now in the FMS must be substituted into this rule in place of the number 6 to indicate that cars now exit the system on the new roller table rather than on roller table 6.

While the addition of a workstation to the system does require modifications to the production system, these modifications are fairly straightforward and don't require any logic changes in the system. In fact this representation provided more flexibility than other options investigated. The production system could be made more flexible by requiring that the user enter the number of workstations in the FMS at system startup. The value supplied by the user could then be used to dynamically generate the initial contents of working memory, and a variable could be used in the production rule fire_t2_output_body_car in place of the hard-coded value.

The storage requirements of this implementation are not unreasonable if the FMS consists of only a few workstations, but if a system is to be developed which includes many workstations, the storage requirements, as well as the execution time required to determine which rule should fire on each cycle of the system could
prove to be a problem. Again, alternate representations are available which will reduce storage requirements.

The Sierra OPS5 expert system development tool was easy to use and was very effective in building this production system. The tools available in OPS5 eased the development process because the steps used in conflict resolution could be viewed as the cars progressed through the system. This clearly showed the value of a rule-based production system as a simulator for studying and analyzing the behavior of an FMS. Bugs or errors were easy to detect and trace as a result of the features available in OPS5, and because the petri net model was available for reference.

The colored petri net model used to develop the rule-based production system proved to be a useful and effective tool, as the petri net clearly defined the requirements of the portion of the FMS modeled.

The three steps outlined in this paper successfully accomplish the objective of this thesis. Though the implementation selected has some limitations, the author believes that the value of this implementation is in the fact that the contents of working memory and the production rules are easily derived, and can be seen to clearly represent the petri net model. As a result of this, the rule-based production system can be easily understood and validated against the petri net model.

The rule-based production system constructed in this thesis could be used as a simulator through which the behavior of the
coordination subsystem could be monitored and analyzed. It could also be used as the control system for this portion of the FMS by interfacing the production system with the transport subsystem and workstation hardware and software.

The implementation of real-time scheduling for the Renault example poses some interesting challenges for future work. Information must be obtained as to the types of failures which might occur in the workstation and the impact each of these failures has on the FMS. In addition, the types of failures that might occur on a roller table must be determined, as well as the methods for recovery from a roller table failure.

Another area deserving further attention is an investigation into the development of rule-based production systems for the petri net addressed in this paper with expert system tools other than OPS5. This investigation should include a comparison of the expert system development tools and how easily they facilitate the development of the production system, as well as comparisons of the rule-based systems actually developed.
APPENDIX A

OPS5 Program Source Code
This program is a Sierra OPSS production system implementation of a petri net model of a
portion of a flexible manufacturing workshop of the Renault Car Company. The purpose of
this portion of the flexible manufacturing workshop is to mastic seal car bodies.

The portion of the flexible manufacturing system being modeled consists of six identical
workstations and a transport subsystem consisting of 6 identical roller tables used to
transport car bodies to and from the workstations. Each workstation has space for one car
body to be worked on and a roller table transfer bench which has two positions, one for
loading a car into the workplace and one for unloading a car from the workplace.

In the system modeled there can be at most two car bodies, bound for the same station, in the
system at any given time. For the sake of simplicity, the system modeled assumes that the
time required to perform each operation in the workstation is identical, and that the time
required to move car bodies from one roller table to the next in the transport subsystem
is identical.

The rules which make up this Sierra OPSS production system were mapped directly from the
petri net contained in Figure 9. For a detailed explanation of the mapping method used,
refer to chapter 6.

**FUSED PETRI NET**

- p1 - remaining cars that can enter the system
- p2 - empty roller tables
- p3 - list of cars moving down roller tables waiting to enter a workstation
- p5 - list of cars moving down roller tables to exit the system
- p6 - empty stations
- p7 - indicates cars in stations being worked on where nothing is on the load or unload
table
- p8 - cars waiting on unload tables to be moved back out to roller tables
- p9 - indicates stations where there is a car in the workplace and on the load table
- p10 - indicates stations where there is a car in the workplace and a car on the unload table

Define elements of working memory used to determine next roller table
(literalize add)

```
  a ; value 1 through 5
  b ; value 2 through 5
```

Define elements of working memory which represent states of the system
(literalize place)

```
  name ; Name of place
  station ; 1 of 6 stations
  table ; 1 of 6 roller tables
  car ; serial number of car
  car2 ; serial number of 2nd car, if any
```
; Rules

; This transition indicates the event where a car is entering the system
\( p \texttt{first1_input_body_car} \)

\[[ \texttt{place \"name s \"station \"et\" car \{sn\}} \texttt{<s>}; \texttt{check if car is waiting to enter system, saving station it is bound for} \]

\[[ \texttt{place \"name p1 \"station \{et\}} \texttt{\{p1\}}; \texttt{check that \{2 cars in system for station \{et\}} \]

\[[ \texttt{place \"name p2 \"table 1} \texttt{\{p2\}}; \texttt{check that table 1 is empty} \]

\rightarrow

\〘 remove \texttt{s}〙

\〘 remove \texttt{p1}〙

\〘 remove \texttt{p2}〙

\〘 make place \"name p3 \"station \{et\} \"table 1} \texttt{\{sn\}}; \texttt{enter record showing there is now a car in the system bound for station \{et\} that currently resides on table 1} \〙

; This transition indicates that a car is ready to exit the system
\( p \texttt{first2_output_body_car} \)

\[[ \texttt{place \"name p5 \"station \{et\} \"table 6} \texttt{\{sn\}} \texttt{\{p5\}}; \texttt{is there a car on \{table 6} that has had all work completed?} \]

\rightarrow

\〘 remove \texttt{p5}〙

\〘 make place \"name p1 \"station \{et\} \〙

\〘 make place \"name p2 \"table 6} \〙

\〘 write \texttt{\texttt{(crlf) Car \{sn\} has been processed.\)}} ;\texttt{write message to screen indicating completion} \〙

; This transition moves a car down the roller table to the desired table (one table at a time)
\( p \texttt{first3_next_table_input} \)

\[[ \texttt{place \"name p3 \"station \{et\} \"table \{tb\} \{st\}} \texttt{\{p3\}}; \texttt{is there a car which needs to be moved to a higher roller table?} \]

\〘 add1 \"a \{tb\} \"b \{tb2\}〙

\[[ \texttt{place \"name p2 \"table \{tb2\}} \texttt{\{p2\}}; \texttt{check that next roller table is empty} \〙
(modify <p2> "table <tb>"

(modify <p3> "table <tb2>"

; This transition moves a car to the next table until it reaches the last table so it may exit
; the system
(p first4_next_table_output

{ (place "name p5 "station <st> "table (tb) ▶ (st)) (p5) } ; is there a car
; which has had all work completed and
; needs to be moved down roller tables
; to table 6 to exit?

{ addl "a (tb) "b (tb2) }  
((place "name p2 "table (tb2) <p2>)

(modify <p2> "table <tb>"

(modify <p5> "table (tb2)"

; This transition loads an empty workstation
(p first3-tl_load_empty_station

{ (place "name p3 "station <st> "table <st> "car (sn) <p3> ) ; is there a car waiting
; on the roller table to enter the
; station?

{ (place "name p6 "station <st>) <p6> }

(remove <p6>)

(remove <p3>)

(make place "name p7 "station <st> "car (sn)"
; create record showing the station now
; has a car being worked on

(make place "name p2 "table <st>)

; place roller table back on the list of
; empty roller tables

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
;This transition places a car on the load table of a station which has a car in it's workplace
  (p first6-tl_load_working_station
   [(place "name p3" station {st} "table {st} "car {sn1}" {p3}) ;is a car waiting on the
    ;roller table to enter the station?
   [(place "name p7" station {st} "car {sn2}" {p7}) ;is a car being worked on in the
    ;station already?
   ->
   (remove {p7}) ;remove record showing car in workplace
   (remove {p3}) ;remove record indicating a car is
    ;waiting on the roller table to enter
   (make place "name p2" table {st}) ;place roller table back on the list of
    ;empty roller tables
   (make place "name p9" station {st} "car {sn1}" "car2 {sn2}" ;create a record
    ;indicating the station now has a car
    ;on the load table and a car in the
    ;workplace
  )
)

;This transition indicates a car in the workplace, nothing on the load and unload tables, work
 ;is completed, and now moving car to unload table
  (p first7-endwork_incomplete_station
   [(place "name p7" station {st} "car {sn}" {p7}) ;has work completed on the car, and
    ;is nothing on the load or unload
    ;table?
   ->
   (remove {p7}) ;remove record above
   (make place "name p8" station {st} "car {sn}" ;create record indicating car has been
    ;moved to the unload table

; Tt8 and tl
;This transition indicates going from a car on the unload table, to a car on the unload table
 ;and another car in the workplace
  (p first8-tl_load_working_station
   [(place "name p3" station {st} "table {st} "car {sn1}" {p3}) ;is a car waiting on the
    ;roller table to enter the station?
   [(place "name p8" station {st} "car {sn2}" {p8}) ;is a car on the unload table?
   ->
   (remove {p8}) ;remove record showing a car only on
    ;the unload table
   (remove {p3}) ;remove record showing a car on the
    ;roller table waiting to get into the
    ;station
   (make place "name p10" station {st} "car {sn1}" "car2 {sn2}" ;create record showing a
    ;car now in the workplace and another
    ;on the unload table and return roller
   (make place "name p2" table {st}) ;table to the available list
  )
)
;This transition moves a car from the unload table and another car from the load table to the workplace.
{p first9_endwork_completes_station
   ;is there a car in the workplace and a car on the load table?
   {place *name p9* station (st) *car (sn1) (sn2)* (p9)}
   ;remove record indicating a car in the workplace and another on load table
   ;create a record showing a car on unload table and another car in workplace
   {remove *p9*}
   {make place *name p10* station (st) *car (sn1) (sn2)* ;is there a car in the workplace and another on unload table?
   {place *name p2* table (st) ;is the roller table available to move the car from the unload table?
   {remove *p10*}
   {remove *p2*}
   ;create record showing a car now ready to be moved down the roller table to exit the system
   ;create record indicating there is now just one car in the workplace
   {make place *name p5* station (st) *table (st) *car (sn2)*
   {make place *name p7* station (st) *car (sn1)*
}

;This transition moves a car from the unload table to the corresponding roller table, while there is a car in the workplace.
{p first10-tu_unload_complete_station
   ;is there a car in the workplace and on the unload table?
   {place *name p10* station (st) *car (sn1) (sn2)* (p10)}
   {place *name p2* table (st) ;is the roller table available to move the car from the unload table?
   {remove *p10*}
   {remove *p2*}
   ;create record showing a car in the workplace and another on the unload table
   ;remove roller table from the available list
   ;create record showing a car now ready to be moved down the roller table to exit the system
   ;create record indicating there is now just one car in the workplace
   {make place *name p5* station (st) *table (st) *car (sn2)*
   {make place *name p7* station (st) *car (sn1)*
}
; Combined til and tu
;This transition moves a car from the unload table to the roller table
(p first-tu unload_incomplete_station
   [(place "name p8 "station (st) "car (sn)) (p8)]; is there just a car on the unload
   [(place "name p2 "table (st) (p2)); is the corresponding roller table
   empty?]
   -->
   (remove (p8)); remove record showing a car on the
   unload table
   (remove (p2)); remove corresponding roller table from
   the available list
   (make place "name p5 "station (st) "table (st) "car (sn)); create record indicating
   car is now ready to exit the system
   (make place "name p6 "station (st)) ; place station back on list of empty
   stations)

; Place P1 is initialized with two tokens for each station. The presence of a colored token,
; is, on place P1 indicates that a car may enter the system bound for station s. Since only two
; cars may be in the system for a given station, P1 initially contains two tokens for each
; station.

; (make place "name pl "station 1)
; (make place "name pl "station 1)
; (make place "name pl "station 2)
; (make place "name pl "station 2)
; (make place "name pl "station 3)
; (make place "name pl "station 3)
; (make place "name pl "station 4)
; (make place "name pl "station 4)
; (make place "name pl "station 5)
; (make place "name pl "station 5)
; (make place "name pl "station 6)
; (make place "name pl "station 6)

; Place P2 is initialized with one token for each roller table in the transport subsystem. The
; presence of a colored token, t, on place P2 indicates that roller table t is empty and a car
; can therefore be placed on that table.

; (make place "name p2 "table 1)
; (make place "name p2 "table 2)
; (make place "name p2 "table 3)
; (make place "name p2 "table 4)
; (make place "name p2 "table 5)
; (make place "name p2 "table 6)
:Place P6 is initialized with one token for each station. The presence of a colored token, s,:
on place P6 indicates that station s is empty, i.e., there are no cars in the workplace nor:
s are any cars on its load or unload table.
;
(make place "name p6 "station 1")
(make place "name p6 "station 2")
(make place "name p6 "station 3")
(make place "name p6 "station 4")
(make place "name p6 "station 5")
(make place "name p6 "station 6")

:In order to make a determination as to whether a car can be moved from one roller table to:
the next, it is necessary to first determine the roller table that the car resides on and:
th en check to be sure that the next roller table is available. In order to accomplish this:
one must be able to perform calculations in the left hand side of a rule, as it is necessary:
to compute one plus the current roller table the car resides on. Since Sierra OSS5 does not:
permit computations on the left hand side of a rule, the following five make statements are:
used to accomplish this task. For example, if it is determined that the car is currently on:
roller table 3 ("a=3") then the next table is table 4 ("b=4")
;
(make addl "a 1 "b 2)
(make addl "a 2 "b 3)
(make addl "a 3 "b 4)
(make addl "a 4 "b 5)
(make addl "a 5 "b 6)

:The following make statements are used to place cars with serial number "car into the system:
bound for stations "station. This will set up the simulation for the cars indicated.
;
(make place "name s "station 2 "car 24)
(make place "name s "station 4 "car 45)
(make place "name s "station 2 "car 26)
(make place "name s "station 5 "car 57)
(make place "name s "station 1 "car 18)
BIBLIOGRAPHY


1987, 543-547.


Martinez, Javier, and Manuel Silva. "A Language for the Description of Concurrent Systems Modeled by Colored Petri Nets: Application to the Control of Flexible Manufacturing Systems." In


